

UNIVERSITÀ
DEGLI STUDI
DI PADOVA



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

SEDE AMMINISTRATIVA: UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

CURRICULUM: SCIENZA E TECNOLOGIA DELL'INFORMAZIONE

CICLO: XXXVIII°

Machine Learning Under Real-World Constraints: Toward Robust, Efficient, and Human-Centric Approaches

DOTTORANDO

Mattia Fanan

Student ID 2079839

SUPERVISORE

Prof. Ruggero Carli

Università degli studi di Padova

CO-SUPERVISORE

Prof. Gian Antonio Susto

Università degli studi di Padova

COORDINATORE

Prof. Fabio Vandin

Università degli studi di Padova

To my family, who have always been by my side, sustaining me with their love and encouraging me through every challenge. This journey would not have been possible without their patience, understanding, and faith in me.

Abstract

Industry 4.0 has profoundly transformed manufacturing and energy systems by introducing digitalization, pervasive sensing technologies, and artificial intelligence. These innovations have enabled unprecedented levels of automation, integration, and optimization across industrial value chains. Nowadays, such systems are no longer isolated technological units but rather interconnected, large-scale, and highly complex infrastructures. They have evolved into critical assets whose failures can lead to severe consequences, including substantial economic losses, environmental degradation, and social disruptions. The complexity and interdependence of these infrastructures make fault prevention and mitigation a matter of strategic importance.

In this context, data-driven predictive maintenance emerges as a cornerstone of modern industrial practices. By exploiting the vast amount of heterogeneous data generated by cyber-physical systems, predictive maintenance enables the early detection of potential faults and anomalies. This proactive approach ensures higher reliability, reduced downtime, and continuity of production processes. Furthermore, it provides industries with the means to optimize resource usage and balance the trade-off between operational efficiency and system safety.

At the same time, the growing demand for sustainability, resilience, and human-centric design has given rise to the paradigm of Industry 5.0. This paradigm does not replace but rather extends the principles of Industry 4.0, by emphasizing the role of technological systems in serving broader societal and environmental goals. Within this framework, predictive maintenance assumes an even more central role, aligned with three key design directions:

- *Efficiency*: to address the rising energy demands induced by digitalization, to minimize waste of resources, and to ensure timely reactions to emerging faults.
- *Robustness*: to guarantee stable and reliable system operation under a variety of operating conditions, including imperfect, uncertain, or dynamic environments.
- *Human-in-the-loop*: to foster trust, accountability, and collaboration between artificial intelligence systems and expert human operators, ensuring that human judgment remains central to decision-making processes.

Machine learning methods have attracted significant attention and witnessed rapid advances in the field of predictive maintenance. Their capability to automatically learn complex patterns from large-scale data makes them a natural solution to address these challenges.

Existing approaches can mainly be categorized into two groups: *general-purpose methods* and *specialized solutions*. General-purpose methods, such as anomaly detection, are designed with flexibility and adaptability in mind, enabling application across a wide variety of domains. These methods typically operate in an unsupervised manner, which is particularly advantageous in scenarios where labeled data are scarce or unavailable. Nevertheless, such generality often comes at the cost of diagnostic depth, limiting their ability to provide actionable insights into the underlying causes of detected anomalies. In contrast, specialized solutions are tailored to specific industrial contexts, often leveraging domain knowledge to enhance diagnostic accuracy and provide richer, context-specific information. While these approaches can facilitate more informed decision-making, they often lack robustness and adaptability when faced with evolving operating conditions or unforeseen scenarios.

Industrial environments add further complexity to this landscape. Obtaining high-quality labeled data is frequently challenging or even impractical, given the rarity of critical failures and the significant costs of controlled fault injection. Moreover, such environments are inherently dynamic, marked by variability in operating conditions, equipment wear, and continuous system evolution. This variability undermines the classical assumption in machine learning that training data are representative of deployment conditions. As a result, model performance tends to degrade over time, requiring frequent retraining, recalibration, or the design of adaptive mechanisms to sustain their effectiveness in the field.

Beyond methodological challenges, regulatory requirements in many domains mandate that safety-critical decisions must involve human oversight. Consequently, the role of machine learning in industrial contexts is increasingly shifting from that of autonomous decision-makers to supportive tools designed to augment human expertise. This shift highlights the necessity of interpretable models capable of communicating their reasoning and outputs in a transparent manner.

In this context, predictive maintenance applied to hydroelectric power plants serves as a representative case study that brings the aforementioned challenges to the extreme, due to the criticality of the infrastructure, the scarcity of fault data, and the dynamicity and complexity of the monitored systems. We worked closely with domain experts from ANDRITZ HYDRO, a global leader in the field of hydroelectric power plants, to deliver with this thesis a set of solutions that are designed for real-world hydroelectric scenarios and validated in general industrial machine learning through proxy tasks. Each of our contributions improves the state-of-the-art in one or more of the following dimensions: *efficiency*, *robustness*, and *human-*

in-the-loop interaction.

Moving from industry-ready applications to more theoretical methodological advances, we can summarize our main contributions as follows:

- A hybrid methodology that combines anomaly detection with interpretability to support human-centric vibration monitoring and root cause analysis in hydroelectric power plants.
- Two novel continual learning methods (SmooER and SmooDER) that exploit data continuity to enable efficient introduction of new classes in behavior-based driver identification, with computational requirements compatible with in-vehicle edge computing. This task is analogous to dynamically introducing new fault types in hydroelectric fault detection.
- RootIF, a feature-evolving anomaly detection method that allows the efficient integration of new features or sensors without the need for historical data, thus reducing retraining vulnerability windows and ensuring compliance with privacy or data storage constraints.
- FLEX-C, a semi-supervised structured ensemble framework for robust fault detection and identification under data scarcity and contamination.

Our dual perspective, i.e., addressing specific hydroelectric scenarios while generalizing them through proxy tasks in industrial machine learning, guided the design of our solutions to be easily adaptable to a wide range of industrial contexts with minimal effort. At the same time, this approach allowed us to overcome the absence of publicly available hydroelectric datasets, which are typically proprietary and confidential, by validating our methods on benchmark datasets that are widely recognized in the literature. Experimental results demonstrate that the proposed approaches consistently match or exceed the performance of traditional state-of-the-art methods under realistic data constraints. Moreover, the integration of interpretability mechanisms effectively supports the work of domain experts, while the computational efficiency of the models enables rapid deployment, even in resource-constrained edge computing environments. Finally, in alignment with the principles of reproducibility and open science, our code has been released to facilitate replication and stimulate further research and development. Links to the corresponding implementations are provided in the respective chapters.

Contents

List of Figures	xi
List of Tables	xiii
List of Algorithms	xvii
List of Acronyms	xvii
1 Introduction	1
1.1 Research Gap and Motivation	3
1.2 Objectives and Scope	3
1.3 Contributions and Publications	4
1.4 Thesis Structure	5
2 Background	7
2.1 Industry 4.0 and 5.0 Frameworks	7
2.1.1 Enabling Technologies of Industry 4.0	8
2.1.2 Research Areas in Industry 4.0	9
2.1.3 Industry 5.0: Human-Centric and Sustainable Paradigm	10
2.2 Predictive Maintenance and Industrial Context	10
2.2.1 Reactive maintenance	11
2.2.2 Preventive maintenance	11
2.2.3 Predictive maintenance	11
2.2.4 Proactive Maintenance	15
2.2.5 Reliability Centered Maintenance	15
2.3 Hydroelectric Power Plant	16
2.3.1 How it works	16
2.3.2 Operating point	17
2.3.3 Signals from the plant	18

CONTENTS

2.4	Machine Learning	21
2.4.1	Supervised Learning	22
2.4.2	Unsupervised Learning	23
2.4.3	Semi-supervised Learning	25
2.4.4	Reinforcement Learning	25
2.4.5	Deep Learning	26
2.4.6	Anomaly Detection	27
2.5	Machine Learning Challenges in Predictive Maintenance	28
2.5.1	Real-time Processing and Infrastructure	28
2.5.2	Data Quality	29
2.5.3	Data Distribution Shift	29
2.5.4	Human-Machine Interaction	31
2.6	Summary	32
3	Research Framework	33
3.1	Research Directions and Proxy Tasks	34
3.1.1	Mapping Projects to Research Directions	35
3.2	General Problem Definition	37
3.3	Synergies	38
4	Anomaly Detection and Human-in-the-Loop for Root Cause Analysis	39
4.1	Background and Related Work	40
4.1.1	Models	40
4.1.2	Interpretability	45
4.1.3	Related Work	48
4.2	Data	49
4.2.1	Data Quality	49
4.2.2	Features	50
4.2.3	Anomalies	51
4.3	Methodology	53
4.3.1	Design Choices	53
4.3.2	Preprocessing	54
4.3.3	Training	54
4.3.4	Evaluation	55
4.4	Experimental Setup	56
4.5	Results and Discussion	57

4.5.1	Isolation Forest	57
4.5.2	Autoencoder	58
4.5.3	KitNet	58
4.5.4	Deep SVDD	59
4.5.5	RSHash	60
4.5.6	SHAP	60
4.6	Summary of Findings	64
5	Continual Learning for Behavior-Based Driver Identification	67
5.1	Proxy Task	69
5.2	Background and Related Work	70
5.2.1	Driver Identification	70
5.2.2	Continual Learning	71
5.3	Methodology	73
5.3.1	Problem Definition	73
5.3.2	Scenarios	74
5.3.3	Methods	77
5.3.4	Prediction Smoothing	78
5.4	Experimental Setup	79
5.4.1	Specifics On The OCSLab Dataset	79
5.4.2	Models And Training	80
5.4.3	Continual Learning Approaches	81
5.4.4	Evaluation Metrics	82
5.5	Results and Discussion	83
5.5.1	Scenario 1 - Two New Drivers	83
5.5.2	Scenario 2 - One New Driver	84
5.5.3	Scenario 3 - Two New Sessions	86
5.5.4	Prediction Smoothing	87
5.5.5	Overall Analysis	87
5.6	Summary of Findings	90
5.6.1	Directions for Future Research	91
6	Feature-Evolving Anomaly Detection	93
6.1	Proxy Task	94
6.2	Background and Related Work	95
6.2.1	Distributional Shifts	96

CONTENTS

6.2.2	Evolving Feature Space	96
6.2.3	xStream	97
6.3	Methodology	99
6.3.1	DIFFI: Depth-based Feature Importance of Isolation Forests	99
6.3.2	Problem Formulation	100
6.3.3	Root-forced Isolation Forest	100
6.3.4	Computational Complexity	103
6.4	Experimental Setup	105
6.4.1	Baseline Methods	105
6.4.2	Datasets	106
6.4.3	Experimental Procedure	108
6.4.4	Evaluation Metrics	110
6.4.5	Ablation Study	113
6.5	Results and Discussion	114
6.5.1	Update Time	115
6.5.2	Inference Time	116
6.5.3	Average Precision	117
6.5.4	Comprehensive Evaluation	119
6.6	Summary of Findings	122
6.6.1	Directions for Future Research	124
7	Robust Fault Detection And Identification	125
7.1	Proxy Task	126
7.2	Background and Related Work	127
7.2.1	Imbalanced Data and Fault Data Scarcity	127
7.2.2	Noisy Labels	129
7.3	Methodology	129
7.3.1	Random Forest	129
7.3.2	Foundational Methods and Adaptation	131
7.3.3	FLEX-C methods	134
7.4	Experimental Setup	141
7.4.1	Data	141
7.4.2	Simulating Nominal Data Contamination	143
7.4.3	Simulating Fault Data Scarcity	144
7.4.4	Models and Baselines	144
7.4.5	Hyperparameters Tuning	146

7.4.6	Evaluation Metrics	147
7.4.7	Implementation Details	148
7.5	Results and Discussion	148
7.5.1	Glass Identification	149
7.5.2	Landsat Satellite	150
7.5.3	Computational Complexity	153
7.6	Summary of Findings	156
7.6.1	Directions for Future Research	157
8	Conclusions and Future Directions	159
8.1	Summary of Contributions	159
8.2	Limitations and Future Directions	161
A	Additional Material for Continual Learning for Behavior-Based Driver Identification	191
A.1	Hyperparameters Tuning	191
A.1.1	ER	191
A.1.2	EWC	192
A.1.3	LwF	194
A.1.4	DER++	194
	Acknowledgments	197

List of Figures

2.1	Representation of when the maintenance strategies are applied with respect to the failure onset (adapted from [36]).	12
2.2	Costs associated with different maintenance strategies, Proactive is often grouped with Predictive (adapted from [38]).	13
2.3	Example of results from a diagnostic Anomaly Detection (AD) approach (adapted from [44]).	14
2.4	Example of results from a prognostic approach (adapted from [51]).	14
2.5	Example of parallel misalignment of the shaft components	20
2.6	Example of angular misalignment of the shaft components	20
2.7	Orbit plot and relative time domain features (images courtesy of [63]).	21
3.1	Mapping of the four projects to the three research directions: efficiency, robustness, and human-in-the-loop.	36
4.1	Examples of representations of the interpretability algorithms.	47
4.2	Representation of frequencies used to compute the main features (image courtesy of [63]).	52
4.3	Anomalies and Training Data Visualization: The graph illustrates the anomaly scores generated by the Autoencoder model, as an example of what to expect in the results. Regions highlighted in red indicate areas with unusual patterns, either identified from company records or subsequently confirmed by them. The green region represents the training set.	53
4.4	Flow chart of the iterative process used to gather the segments of the data that are to be considered as anomalies	56
4.5	Plot of the anomaly score from the iForest model.	57
4.6	Plot of the anomaly score from the Autoencoder model.	58
4.7	Plot of the anomaly score from the KitNet model.	59
4.8	Plot of the anomaly score from the Deep SVDD model.	59

LIST OF FIGURES

4.9	Plot of the anomaly score from the RSHash model.	60
4.10	SHAP output before January 30th - relevant features: gap_bearing_rX (orange line '—') and the 1x_magn_bearing_aX (maroon line '—').	62
4.11	SHAP January 30th - bearing's relevant features for the first peak are asynchronous components in X direction (dark blue line '—'), asynchronous components in Y direction (dark yellow line '—') for shaft-bearing displacement. Note that the synchronous spectral component for bearing velocity in the X direction is higher than the ones represented.	63
4.12	SHAP output with unconnected sensor - relevant features: gap on X direction (orange line '—'), gap on Y direction (salmon line '—'), crest on X direction (green line '—'), crest on Y direction (light pink line '—') and synchronous component's phase on X direction (dark magenta line '—'). Where all of them represent the shaft-bearing displacement.	64
4.13	SHAP output on the new anomaly - relevant features: sub-synchronous spectral components (teal line '—'), 2x synchronous spectral component (light green line '—'), synchronous spectral component (blue line '—'), vortex rope's spectral component (olive line '—'), asynchronous spectral components (indigo violet line '—') and Root Mean Square (RMS) (orange-yellow line '—') all of them in the structure along the X direction.	65
5.1	General representation of the experimental setup. Since the goal is to learn new tasks as they arrive while retaining knowledge from previous ones, the test set for each task consists of all the test data from drivers whose labels already appeared.	69
5.2	Task sequence representation for Scenario 1. Given ten drivers, each task contains information about two new drivers, for a total of five tasks.	75
5.3	Task sequence representation for Scenario 2. A single driver is added to the system each time (except for task 1).	75
5.4	Task sequence representation for Scenario 3. The data is divided into drive sessions (there are two of them for each driver) and each task contains the data of two sessions. As a result, the next task can either introduce new drivers or expand the knowledge of previously encountered ones.	76
5.5	Results for Scenario 1.	84
5.6	Results for Scenario 2.	85
5.7	Results for Scenario 3.	86

5.8	Comparison between DER++ and SmooDER. The test set windows are organized by their drive sessions index and driver ID, with SmooDER successfully suppressing most classification errors.	88
5.9	Training time per task for each method and baseline in Scenario 3. Cumulative training time increases linearly as more tasks are added since the datasets are concatenated. Fine-tuning and LwF have negligible or no computational overhead. EWC computational time grows since by implementation details each task contributes independently to the regularization. DER++, ER, SmooER, and SmooDER go to saturation when the memory is filled for the first time. The overhead of SmooDER over DER++ is negligible (same for SmooER over ER).	90
6.1	Convergence of A_∞^α for input sequence $(5, 2+6^{-k-1}, \dots)$ with $k \in \{1, \dots, 1000 \text{ or } 100000\}$. Truncation error causes the deviation from the true limit at $x = 1$. However, this error becomes negligible when early or convergence terms are dominant.	113
6.2	Update times for each RootIF configuration and baselines on the Mammography dataset, with 100 new trees trained per update. The x-axis displays dual metrics: the number of features (top) and the cumulative size of the training dataset (bottom) for each experience.	115
6.3	Evolution of the update time for each RootIF configuration and baselines on the Mammography dataset with different numbers of new trees trained at each update.	116
6.4	Inference time of each configuration of RootIF on the Mammography dataset with 100 new trees trained at each update.	117
6.5	Evolution of the inference time of RootIF on the Mammography dataset with different numbers of new trees trained at each update.	118
6.6	Evolution of the AP of RootIF on the Pima dataset with different numbers of new trees trained at each update.	119
6.7	Examples of our model outperforming and being outperformed by xStream.	121
6.8	Comprehensive comparison of update and inference time on the Mammography dataset.	121
7.1	Accuracy performance of models on the Glass Identification dataset. The plot is presented on a logarithmic scale to facilitate interpretation.	149
7.2	Accuracy performance of models on the Landsat Satellite dataset. The plot is presented on a logarithmic scale to facilitate interpretation.	151

LIST OF FIGURES

A.1	Results of the <i>Memory Size</i> parameter search for the ER method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.	192
A.2	Results of the <i>Replay Ratio</i> parameter search for the ER method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.	192
A.3	Results of the λ parameter search for the EWC method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.	193
A.4	Results of the λ parameter search for the LwF method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.	194
A.5	Results of the α parameter search for the DER++ method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.	195
A.6	Results of the β parameter search for the DER++ method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.	195
A.7	Results of the <i>Memory Size</i> parameter search for the DER++ method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.	196

List of Tables

4.1	Table with the number of samples for each OS.	50
4.2	Elapsed times for training and execution with a training set of 37668 points and a total dataset of 61314 points (average of 5 trials).	60
5.1	Count of windows in training and test set for each driver's drive session. . . .	81
5.2	Performance for each scenario and for each CL technique. The best accuracy for each scenario is highlighted in bold.	85
5.3	Performance before and after the prediction smoothing for each scenario and for each CL technique.	87
6.1	Statistics of ODDS datasets	107
6.2	Settings used in our experiments	109
6.3	Average RWP across three experiences (best in bold), we report the baselines and our model with all the components active. We also report the performance (Abl.) and name (Abl. Name) of the best model from our ablation study. . . .	120
6.4	Average improvement in terms of RWP across the three experiences and four component combinations. The values represent the difference in RWP when the component is enabled compared to when it is disabled (e.g. "force_diffi - force" is one of the four terms to average for diffi), with positive values indicating an improvement.	123
6.5	Average AP difference between removing the first and last two features in TreeSHAP order of importance. Positive values indicate that certain features are more discriminative for anomaly detection.	123
7.1	Classes with meaning and number of samples in the Statlog dataset.	142
7.2	Glass types and number of samples in the Glass Identification dataset. Float processing is a glass manufacturing technique.	143

LIST OF TABLES

7.3	Average confusion matrices across all seeds for the FLEX-C Labels and FLEX-C Centroids models on Glass Identification dataset with one SPF. The diagonal cells (green) represent correct classifications, while the off-diagonal cells (red) represent misclassifications.	150
7.4	Average confusion matrices across all seeds for the FLEX-C Centroids and FLEX-C Centroids v2 models on Landsat Satellite dataset with 1 SPF. The diagonal cells (green) represent correct classifications, while the off-diagonal cells (red) represent misclassifications.	154
7.5	Average confusion matrices across all seeds for the FLEX-C Centroids and FLEX-C Centroids v2 models on Landsat Satellite with 438 SPF. The diagonal cells (green) represent correct classifications, while the off-diagonal cells (red) represent misclassifications.	154
7.6	Comparison of theoretical time complexities for training and inference across different models. The variables used are defined below.	155

List of Algorithms

1	Root-forced Isolation Tree Construction	102
2	Adaptive anomaly score computation	104
3	Update Method	135
4	Predict Labels	136
5	Incremental Centroid Update	138
6	Predict Function for FLEX-C Centroids	139

List of Acronyms

AI Artificial Intelligence

AL Active Learning

ALIF Active Learning Isolation Forest

BALIF Bayesian Active Learning Isolation Forest

AD Anomaly Detection

AP Average Precision

CAN Controller Area Network

CNN Convolutional Neural Network

CL Continual Learning

CPS Cyber-Physical Systems

DIFFI Depth-based Feature Importance of Isolation Forest

DL Deep Learning

FDI Fault Detection and Identification

FeV Feature-Evolving

FLEX-C Forest of Local EXpert Classifiers

IoT Internet of Things

iForest Isolation Forest

iTree Isolation Tree

LIST OF ALGORITHMS

GPS Global Positioning System

KDE Kernel Density Estimation

KNN K-Nearest Neighbors

LSTM Long Short-Term Memory

LIME Local Interpretable Model-Agnostic Explanations

HPP Hydroelectric Power Plant

ML Machine Learning

MSE Mean Squared Error

NN Neural Network

OBD-II On-Board Diagnostics

OS Operating State

OCSVM One-Class Support Vector Machine

PCA Principal Component Analysis

PdM Predictive Maintenance

RF Random Forest

RCA Root Cause Analysis

RCM Reliability Centered Maintenance

RUL Remaining Useful Life

RNN Recurrent Neural Network

RMS Root Mean Square

RL Reinforcement Learning

RSI Rotor-Stator Interaction

RWP Resource-Weighted Performance

RootIF Root-forced Isolation Forest

RSHash Randomized Subspace Hashing

SHAP SHapley Additive exPlanations

SOTA State-Of-The-Art

SPF Samples Per Fault

SOM Self-Organizing Map

Deep SVDD Deep Support Vector Data Description

SVM Support Vector Machine

XAI eXplainable Artificial Intelligence

LwF Learning without Forgetting

EWC Elastic Weight Consolidation

ER Experience Replay

DER Dark Experience Replay

DER++ Dark Experience Replay++

SmooER Smooth Experience Replay

SmooDER Smooth Dark Experience Replay



Introduction

Over the last decade, Industry 4.0 has been driven by digitalization and Artificial Intelligence (AI) technologies aimed at improving efficiency and production flexibility [1]. Its core enablers include the Internet of Things (IoT), big data analytics, and cloud computing [2]. However, as noted in [3], the initial guiding principles of fairness and sustainability have often been overshadowed by the pursuit of technological progress. Industry 5.0 addresses this imbalance by building upon the foundations of Industry 4.0 while extending them to emphasize sustainability, human-centricity, and resilience. Rather than a chronological successor, it should be seen as a complementary framework that responds to evolving societal needs while reinforcing established industrial trends. In this way, the current industrial landscape is defined by the coexistence of the two paradigms, together with hybrid forms that combine elements of both [4].

Within this broader context, Predictive Maintenance (PdM) has emerged as a central application. It is defined as the use of data-driven approaches to anticipate equipment failures so that maintenance can be performed proactively and at the right moment [5]. By leveraging advanced data analytics, Machine Learning (ML), and Deep Learning (DL), PdM enables operators to monitor the condition of critical assets, anticipate malfunctions, and optimize intervention schedules. This is especially true in the energy sector where reliability and efficiency are critical for operational continuity.

These challenges are illustrated by Hydroelectric Power Plants (HPPs). In 2023, the net hydroelectric contribution to electricity generation in Europe averaged 12.1%, with countries such as Norway relying on hydroelectric power for as much as 89% of their total production [6]. Beyond being a major renewable source, hydroelectric power plays a strategic role in energy storage. By using surplus electricity to pump water uphill into reservoirs and later

releasing it to generate power, it functions as a large-scale natural battery [7]. This capability is crucial for balancing the variability of solar and wind generation and contributes directly to energy security [8, 9]. In this context, PdM supports both the reliability and efficiency of HPP, ensuring uninterrupted service and reinforcing the stability of the energy system.

Despite its promise, the application of ML to PdM faces significant challenges in industrial settings. The first major challenge is data quality and availability. While raw sensor data is typically abundant, many ML approaches rely on labeled data for effective training. Creating labeled datasets requires expert knowledge and manual effort, which are costly and time-intensive, resulting in relatively small training datasets. In practice, labels are often generated in an event-driven manner, created only when a fault is detected, investigated, and documented in company archives. This process frequently leads to incomplete or noisy annotations, where boundary cases may be misclassified, and hidden faults may remain unnoticed during what is considered normal operation. Both limited dataset size and noisy labels hinder model training, reducing predictive accuracy [10, 11].

Another difficulty lies in the sheer volume of raw data generated by industrial systems. Efficient ML methods are required to process this information, often in conjunction with edge computing solutions. Moving computation closer to the data source reduces latency, improves responsiveness, and addresses privacy concerns by avoiding centralization of sensitive operational data [12].

Also, it must be noted that industrial environments are inherently dynamic, and models must therefore adapt to new conditions with minimal delay in order to reduce vulnerability periods during which faults may go undetected. Such adaptation can be achieved through Continual Learning (CL), which enables models to continuously integrate incoming data to address concept drift and learn new fault types without discarding previously acquired knowledge [13]. In other situations, the challenge is linked to the evolution of input signals, for instance when additional sensors or features are introduced. These scenarios require models that can effectively operate under Feature-Evolving (FeV) conditions [14].

Finally, the deployment of AI in safety-critical domains such as PdM for HPP is subject to strict regulation. The European AI Act explicitly requires that human operators supervise AI systems and retain the authority to override automated decisions when necessary [15]. This principle of human-in-the-loop resonates strongly with the vision of Industry 5.0, which emphasizes collaborative interaction between humans and AI. In this setting, automation provides efficiency while human judgment safeguards safety, accountability, and trust.

1.1 RESEARCH GAP AND MOTIVATION

The use of ML in PdM for industrial applications has attracted increasing attention in recent years [16], with existing methods largely falling into two categories.

The first category consists of general-purpose approaches, often adapted from broader domains to industrial contexts. AD is a prominent example: it is widely adopted because of its unsupervised nature, which removes the need for labeled data, and because of its resilience to imbalanced datasets. Although these methods are flexible and straightforward to deploy, they provide only limited insight.

The second category focuses on specialized solutions designed for particular applications. These methods tend to deliver richer diagnostic information, but this comes at the expense of generality and robustness. As a result, they are often difficult to apply in the dynamic and diverse conditions typical of industrial environments. Classification models, for instance, may yield more informative outputs but are frequently constrained by limited data and unreliable labels.

Another important challenge is the interaction between human personnel and AI systems. Limited interoperability reduces trust, slows adoption, and undermines both efficiency and safety [17].

Taken together, these observations point to a gap between the two extremes. What is needed are approaches that combine the efficiency, robustness, and ease of deployment characteristic of general-purpose methods with the clear insights delivered by specialized ones.

1.2 OBJECTIVES AND SCOPE

The main goal of this thesis is to develop a suite of ML methods for PdM in hydroelectric systems that complement each other, and are easy to adapt and deploy, while ensuring broad applicability across industrial domains. Considerations on the balance between domain-specific and general-purpose through the use of proxy tasks are further discussed in Chapter 3.

To make PdM methods truly applicable and valuable in industry, this work focuses on the improvements in three key directions:

- *Efficiency*: methods that adapt to changing deployment environments without requiring extensive retraining, minimizing computational and data requirements.

1.3. CONTRIBUTIONS AND PUBLICATIONS

- *Robustness*: models that sustain performance in the presence of common industrial challenges like unbalanced data or unreliable labels.
- *Human-in-the-loop*: approaches that integrate a preferably bidirectional interaction between human operators and AI systems.

1.3 CONTRIBUTIONS AND PUBLICATIONS

The contributions of this Thesis are reflected in the following published and submitted works:

- *Anomaly Detection for Hydroelectric Power Plants: a Machine Learning-based Approach* [18], published at the 2023 IEEE 21st International Conference on Industrial Informatics (INDIN) and discussed in Chapter 4. This work compares state-of-the-art anomaly detection methods in the context of predictive maintenance in hydroelectric power plants and proposes a methodology that combines anomaly detection with interpretability to support fault detection and root cause analysis. Using real data from hydroelectric facilities, we showed that Autoencoders provide the best balance of performance, accuracy, and output readability, while SHapley Additive exPlanations (SHAP) is effective in pointing the expert towards the features that simplify the identification of the fault's causes.
- *Continual Learning for Behavior-based Driver Identification* [19], published in *Engineering Applications of Artificial Intelligence* and discussed in Chapter 5. This work investigates the driver identification problem within a continual learning framework. We designed three progressively challenging scenarios on the OCSLab dataset to mimic real-world deployment conditions and evaluated state-of-the-art continual learning methods in these settings. We also proposed two novel approaches, Smooth Experience Replay (SmooER) and Smooth Dark Experience Replay (SmooDER), which exploit the natural continuity of driver identity and achieve superior results compared to existing techniques, demonstrating feasibility for in-vehicle deployment.
- *Feature-Evolving Anomaly Detection via Root-forced Isolation Forest* [20], submitted to *Engineering Applications of Artificial Intelligence* and discussed in Chapter 6. This work introduces Root-forced Isolation Forest (RootIF), a feature-evolving anomaly detection method based on Isolation Forest. It allows the incremental integration of new sensors or features with minimal new data requirements, which reduces time to deployment

and consequently shortens the system’s vulnerability window, all while avoiding the need for access to historical data and thereby minimizing privacy concerns. We compared it with the state-of-the-art method xStream and demonstrated competitive or superior performance across several public datasets.

- *FLEX-C: A Robust Framework for Fault Detection and Identification under Scarce and Contaminated Data* [21], submitted to *Engineering Applications of Artificial Intelligence* and discussed in Chapter 7. This work introduces Forest of Local EXpert Classifiers, a semi-supervised structured ensemble framework for fault detection and identification designed to be robust when fault data is scarce and normal operation data is abundant but potentially contaminated. We developed two variants, FLEX-C Labels and FLEX-C Centroids, and demonstrated through systematic comparisons with Random Forest and other baselines that FLEX-C can match or surpass the performance of a Random Forest trained on clean and balanced data within realistically obtainable dataset sizes.

In addition, the following publication was not included in this thesis due to limited synergy with the main contributions and the use of proprietary datasets that limits reproducibility and future improvements.

- [22] *Time Series Forecasting to Detect Anomalous Behavior in Multiphase Flow Meters*, published at the 2024 IEEE 8th Forum on Research and Technologies for Society and Industry Innovation (RTSI).

1.4 THESIS STRUCTURE

The thesis is organized as follows:

- **Chapter 2: Background and Literature Review** introduces the foundations required to understand this thesis. It first outlines the broader context of Industry 4.0 and 5.0, followed by a concise overview of hydroelectric power plants to help readers outside the field grasp the application setting and the content of Chapter 4 as well as the proxy tasks. A lightweight taxonomy of maintenance approaches is then presented, together with an introduction to machine learning and a review of its applications in industry. Anomaly detection is discussed as a recurring theme across the contributions. The chapter concludes with an overview of the main challenges in applying machine learning to industrial contexts.

1.4. THESIS STRUCTURE

- **Chapter 3: Research Framework** presents the overall framework, defines the general problem, and describes the mapping between proxy tasks and hydroelectric scenarios, highlighting the synergies among the contributions.
- **Chapter 4: Anomaly Detection and Human-in-the-Loop for Root Cause Analysis** presents our first contribution, which integrates anomaly detection with interpretability for fault detection and root cause analysis in hydroelectric predictive maintenance.
- **Chapter 5: Continual Learning for Behavior-Based Driver Identification** presents our second contribution, which investigates the application of CL to the behavior-based driver identification problem, including our novel approaches (SmooER and SmooDER).
- **Chapter 6: Feature-Evolving Anomaly Detection** presents our third contribution, which proposes Root-forced Isolation Forest, an extension of Isolation Forest that enables efficient incremental integration of new sensors or features.
- **Chapter 7: Robust Fault Detection And Identification** presents our fourth contribution, which introduces FLEX-C, a semi-supervised structured ensemble framework for fault detection and identification under scarce and contaminated data, with two variants (FLEX-C Labels and FLEX-C Centroids).
- **Chapter 8: Conclusions and Future Directions** summarizes the contributions, implications, and potential directions for future research.

2

Background

This chapter provides the essential background and context for the thesis.

Section 2.1 introduces the concepts of Industry 4.0 and 5.0. Section 2.2 then reviews key approaches to PdM in industrial contexts. Section 2.3 follows with an introduction to HPP, which provides the application context that motivated the development of the methods in this thesis. This section includes a basic explanation of its principles and the type of data it generates, helping readers from various backgrounds follow the content in Chapter 4 and the proxy tasks discussed in Chapters 5, 6, and 7.

Section 2.4 introduces ML, using the simplified taxonomy from [23], and also presents AD, a concept that is central to many of the thesis contributions.

Section 2.5 highlights the main challenges of applying ML to PdM, providing motivation for the thesis contributions.

The chapter closes with a summary.

2.1 INDUSTRY 4.0 AND 5.0 FRAMEWORKS

The industrial landscape has undergone a series of deep transformations since the First Industrial Revolution. The mechanization of production in the late 18th century introduced steam power and mechanized textile manufacturing, while the second industrial revolution brought mass production, electrification, and assembly lines. The third industrial revolution, beginning in the latter half of the 20th century, was driven by automation, electronics, and the early adoption of information and communication technologies. Each phase of industrial evolution has progressively enhanced productivity, efficiency, and scale, laying

2.1. INDUSTRY 4.0 AND 5.0 FRAMEWORKS

the foundation for the paradigms of Industry 4.0 and Industry 5.0 [24].

Industry 4.0, often referred to as the Fourth Industrial Revolution, represents the convergence of digitalization, automation, and advanced connectivity within manufacturing and service systems. It emphasizes the integration of Cyber-Physical Systems (CPS), IoT, and data-driven intelligence to create smart factories and interconnected industrial environments [1, 25]. This enables real-time monitoring, adaptive decision-making, and autonomous operations across the production lifecycle. Industry 4.0 thus extends beyond traditional automation by embedding intelligence into physical assets, creating value chains that are more flexible, resilient, and responsive to dynamic market demands.

2.1.1 ENABLING TECHNOLOGIES OF INDUSTRY 4.0

The foundation of Industry 4.0 is built on a set of enabling technologies that transform traditional manufacturing systems into interconnected and intelligent environments. The most prominent among these include:

- **Internet of things:** A network of physical objects (such as devices, vehicles, buildings, and other items) that are equipped with electronics, sensors, and software. These components enable the objects to collect and share data over existing networks. By allowing objects to be monitored and controlled remotely, Internet of Things creates opportunities to connect the physical world with digital systems, leading to greater efficiency and accuracy [26].
- **Big data analytics:** A set of tools and methods for managing, processing, and analyzing data across the five key dimensions (volume, variety, velocity, veracity, and value). These techniques generate actionable insights that support continuous value creation, performance measurement, and competitive advantage [27].
- **Cyber physical systems:** Systems that integrate the dynamics of physical processes with computation and communication, providing unified abstractions along with modeling, design, and analysis techniques for the combined whole. [28].
- **Cloud computing:** A model that provides ubiquitous, convenient, and on-demand access to a shared pool of configurable computing resources such as networks, servers, storage, applications, and services. These resources can be rapidly provisioned and released with minimal management effort, as defined by the NIST framework [29].

In addition to the core technologies, several other enablers are essential to the Industry 4.0 ecosystem. These include **Artificial Intelligence (AI)** and **Machine Learning (ML)**

for predictive analytics and decision-making, **advanced robotics** for flexible and precise operations, **blockchain** for secure and decentralized transactions, **additive manufacturing** for rapid and customizable production, **augmented reality** for enhanced human-machine interaction, **cybersecurity** for protecting digital assets and infrastructure, and **digital twin** technology for creating virtual representations of physical systems that support monitoring and optimization.

Together, these technologies form the backbone of the **smart factory**, characterized by self-optimization, real-time adaptability, and seamless end-to-end digital integration across the entire value chain [2].

2.1.2 RESEARCH AREAS IN INDUSTRY 4.0

The widespread adoption of Industry 4.0 technologies has created numerous research areas. Among the most active are:

- **Predictive maintenance:** Analyzing sensor data with statistical techniques, expert system reasoning, or machine learning methods to forecast equipment failures before they occur, thereby reducing downtime and extending asset lifespan.
- **Fault detection:** Applying advanced algorithms to identify anomalies in real time, improving operational safety and reliability.
- **Production optimization:** Leveraging data-driven approaches and simulation models to increase throughput, minimize waste, and balance resource utilization.
- **Quality control:** Integrating vision systems and sensor measurements with techniques such as automated visual inspection, multivariate statistical monitoring, and data driven anomaly detection to detect defects early in the production process, improving consistency and lowering costs.
- **Supply chain optimization:** Employing big data analytics and real-time tracking to streamline logistics, reduce bottlenecks, and build resilience against disruptions.
- **Asset management:** Combining digital twins and monitoring frameworks to provide greater visibility into equipment health and overall system performance.

Together, these research directions highlight how Industry 4.0 is driving innovation across every stage of industrial practice, with PdM and fault detection being the core focus of this work.

2.2. PREDICTIVE MAINTENANCE AND INDUSTRIAL CONTEXT

2.1.3 INDUSTRY 5.0: HUMAN-CENTRIC AND SUSTAINABLE PARADIGM

While Industry 4.0 has significantly advanced automation, connectivity, and data-driven decision-making, its implementation has also revealed several limitations. Heavy reliance on automation and digital technologies can lead to workforce displacement, increased energy consumption, and concerns regarding data security and system complexity. These shortcomings have given rise to the concept of Industry 5.0, which seeks to *complement* technological progress with human-centric values and sustainability considerations [3].

Industry 5.0 emphasizes the collaboration between humans and intelligent machines, highlighting the role of human creativity, critical thinking, and problem-solving in harmony with advanced automation. Instead of replacing the workforce, the focus shifts toward empowering operators through collaborative robots (cobots), decision-support systems, and human-in-the-loop design strategies [30].

In parallel, sustainability has become a central theme of Industry 5.0. This includes the integration of green manufacturing practices, energy-efficient processes, and circular economy principles to reduce environmental impact [31]. The human-centric and sustainable orientation of Industry 5.0 therefore marks a transition from efficiency-driven automation to a balanced framework that values societal well-being, environmental stewardship, and resilient innovation.

2.2 PREDICTIVE MAINTENANCE AND INDUSTRIAL CONTEXT

Maintenance plays a crucial role in ensuring the profitability and competitiveness of modern companies. It safeguards system longevity (asset management), safety, availability, efficiency, and quality [32]. According to ISO 14224 (2004), maintenance is officially defined as *the combination of technical and associated administrative actions intended to retain an item or system in, or restore it to, a state in which it can perform its required function*.

From a company's perspective, maintenance often represents a significant share of operating costs. In [33] the authors report that maintenance-related expenses can account for as little as 15% or as much as 70% of total production costs. Beyond cost, plant availability is another critical concern: depending on the component to be repaired, maintenance can require long downtimes and even a complete system shutdown. To address these issues, various strategies have been developed over time with the dual aim of reducing costs and increasing system availability.

Terminology related to maintenance strategies is often inconsistent in the literature. To establish a common ground, we follow [34], which clarifies the nomenclature most widely

adopted in industry today. Accordingly, maintenance approaches are classified into four categories: Reactive, Preventive, Predictive, and Proactive. In the following sections, these strategies will be explained in detail, along with Reliability Centered Maintenance (RCM), which is often considered separately since it combines different strategies for individual components rather than applying a single strategy across the entire system.

2.2.1 REACTIVE MAINTENANCE

Reactive maintenance, also known as run-to-failure, failure-based, or corrective maintenance, is historically the first strategy adopted in industry [35]. As shown in Figure 2.1a, it is also the simplest, since maintenance is carried out only after a component fails or stops functioning properly.

The main advantage of this approach lies in its simplicity: it only requires detecting failures after they occurred. However, the drawbacks are considerable. Failures can cause unpredictable production interruptions and may pose safety risks to both the system and personnel. These reasons make reactive maintenance only suitable when failures are isolated and the associated risks can be managed effectively.

2.2.2 PREVENTIVE MAINTENANCE

Preventive maintenance (PvM), also known as scheduled or interval-based maintenance, represents a more advanced approach compared to reactive maintenance. As shown in Figure 2.1b, its goal is to prevent unplanned system failures, which in many contexts are unacceptable due to the high costs of sudden production interruptions. The principle behind this strategy is to determine time intervals during which components are likely to perform reliably. A common example is the periodic oil change in cars, carried out after a certain mileage even if the oil could still function for some time.

The advantage of preventive maintenance lies in its ability to greatly reduce the likelihood of unexpected failures. However, it also has drawbacks. Determining appropriate maintenance intervals requires a thorough system analysis, and components are often replaced before the end of their useful life. This can result in additional costs for spare parts and labor.

2.2.3 PREDICTIVE MAINTENANCE

Predictive Maintenance (PdM), also known as condition-based maintenance, is a strategy focused on monitoring and analyzing system conditions to ensure the optimal interval between repairs and minimize the cost of unscheduled outages [37]. As shown in Figure

2.2. PREDICTIVE MAINTENANCE AND INDUSTRIAL CONTEXT

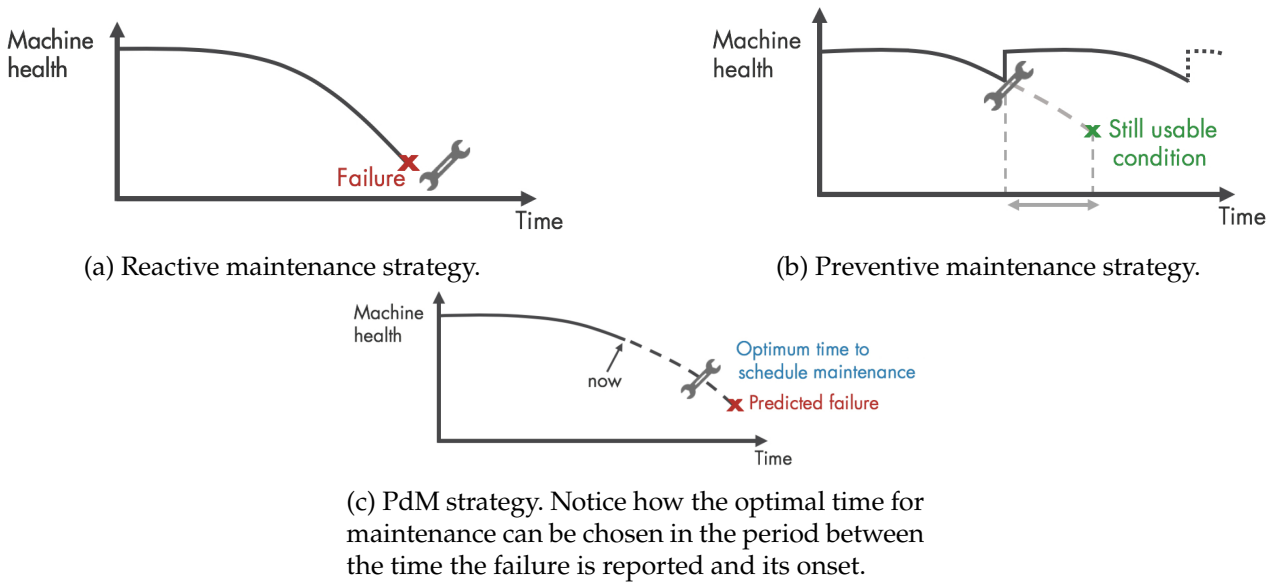


Figure 2.1: Representation of when the maintenance strategies are applied with respect to the failure onset (adapted from [36]).

2.1c, maintenance is carried out when measurable indicators suggest that a component is approaching the end of its reliable service life.

A practical example is provided in [34]: brake pads in cars are typically replaced when their thickness drops below a threshold. The monitored condition is the pad thickness, which reflects accumulated wear. Since fully worn pads cannot guarantee braking, a threshold is set to ensure that maintenance can be scheduled in advance.

As illustrated in Figure 2.2, PdM strikes a better balance than reactive and preventive strategies. It allows users to exploit a component's lifespan while reducing the likelihood of unexpected failures. Once a key trend indicator triggers an alarm, maintenance can be scheduled at the most convenient time, minimizing production losses. Costs are comparable to run-to-failure strategies, since maintenance is only performed when needed, while reliability, availability, and profitability are significantly improved.

The main drawback of PdM lies in the complexity of acquiring, storing, and managing the large quantities of data it requires. Companies must invest in infrastructure or outsource these tasks. Nonetheless, this abundance of data enables remote centralized control and supports the application of AI models to continuously optimize maintenance policies.

Over time, numerous methods have been developed for data-driven PdM. In the literature, State-Of-The-Art (SOTA) approaches are generally classified into two categories: diagnostics and prognostics.

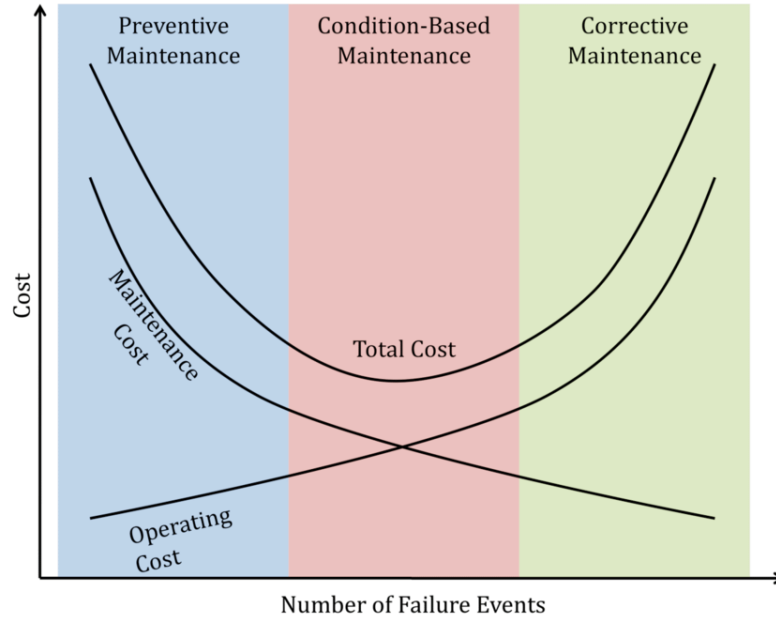


Figure 2.2: Costs associated with different maintenance strategies, Proactive is often grouped with Predictive (adapted from [38]).

DIAGNOSTICS

The goal of diagnostic approaches is to characterize the current state of a system. Common methods for this purpose include classification models that directly distinguish between different fault types, such as our proposed Forest of Local EXpert Classifiers (FLEX-C) presented in Chapter 7. Another family of techniques is AD, which includes methods like Isolation Forest (iForest) and its variants [39], Self-Organizing Map (SOM) [40], and Neural Network (NN)-based models such as Autoencoders [41]. These approaches are often used to highlight faulty regions in the data, as illustrated in Figure 2.3.

This approach offers the advantage of capturing correlations between features directly from data, which enables the detection of complex faults that depend on many variables [42]. In the presence of a fault, it can also directly pinpoint its causes. The main drawback is that the timing of maintenance must be inferred through additional strategies such as those proposed in [43].

PROGNOSTICS

In contrast, a prognostic approach aims to predict the future evolution of a system, as illustrated in Figure 2.4, often in terms of the expected time to the next fault, as in Remaining Useful Life (RUL), or in terms of the evolution of the fault probability, as in Risk Estimation.

2.2. PREDICTIVE MAINTENANCE AND INDUSTRIAL CONTEXT

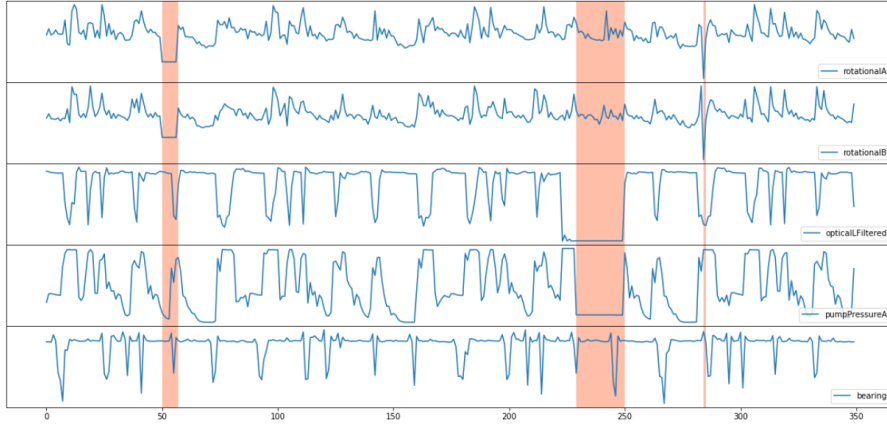


Figure 2.3: Example of results from a diagnostic AD approach (adapted from [44]).

These approaches typically focus on individual components or features [45]. An additional layer is then required to aggregate results at the system level [46]. The main advantage of this method is its straightforward interpretation: it essentially provides a countdown to failure, simplifying maintenance planning. However, the challenge lies in the high complexity of the task and the limited availability of high quality data. As a result, models often suffer from reduced precision, which accumulates across the system.

A wide variety of algorithms can be employed for RUL estimation. Traditional physics based methods include finite element modeling and experimental modal analysis [47]. Data driven techniques range from SOM and back propagation NN [48] to Recurrent Neural Network (RNN) and neuro-fuzzy systems [49]. More recently, Transformer based architectures have also gained popularity [50].

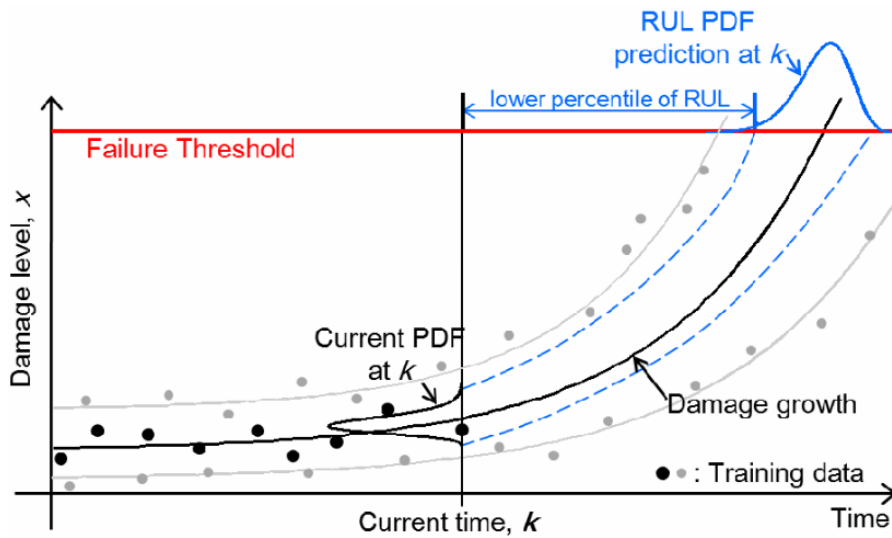


Figure 2.4: Example of results from a prognostic approach (adapted from [51]).

2.2.4 PROACTIVE MAINTENANCE

Proactive maintenance (PaM) can be seen as an evolution of PdM. As explained in [52], once personnel gained the expertise to interpret PdM data, they began not only to respond to signs of degradation but also to identify and eliminate the root causes of potential failures. In some sources, such as [37], proactive and PdM are merged under a single category.

To clarify the concept, consider two examples from [34] and [53]. In car maintenance, ensuring tires are inflated to the correct pressure and aligned properly reduces uneven wear and extends tire life. When a driver visits a service center to adjust tire pressure and alignment, proactive maintenance is being performed. Similarly, in systems where fluids are essential for operation, contamination can accelerate component wear. By monitoring fluid contamination and taking corrective actions, such as improving filtration or replacing the fluid when contamination reaches a critical level, companies prevent premature degradation.

The strength of proactive maintenance lies in enhancing reliability beyond what PdM can achieve. However, this comes at the cost of requiring greater expertise and resources. Personnel must be capable of applying advanced methods such as Root Cause Analysis (RCA), failure modes and effects analysis, and fault tree analysis to identify and mitigate the underlying causes of system degradation.

2.2.5 RELIABILITY CENTERED MAINTENANCE

Reliability Centered Maintenance (RCM) is defined in [54] as *a process used to determine what must be done to ensure that any physical asset continues to do whatever its users want it to do in its present operating context*. Unlike the previous strategies, RCM does not prescribe a single approach for the entire system. Instead, it assigns different maintenance strategies to individual components depending on their criticality and operational requirements.

A practical example is described in [34]. In the case of car maintenance, different components fall under different strategies: brake pads are typically managed with PdM, tires with proactive maintenance, and engine oil with preventive maintenance, while light bulbs are usually replaced only when they fail, following a reactive approach. Taken together, these strategies ensure that the car, as a system operates under RCM.

The main advantage of RCM is its flexibility. By tailoring the strategy to each component, it combines the reliability and cost benefits of advanced approaches with the simplicity of reactive ones where appropriate. This allows companies to reduce unnecessary complexity and costs while still maintaining system performance and safety.

2.3 HYDROELECTRIC POWER PLANT

A HPP is a complex system that transforms the potential energy of water into electricity. Since the focus of this work is more general, the goal here is not to provide a full technical description, but rather a short overview to give readers the essential background to understand the RCA of Chapter 4 and the correlation with proxy tasks in Chapters 5, 6 and 7. In the following we will give an overview of how HPPs operate and which types of signals are most relevant for detecting faults.

There are different types of HPPs, depending on the way water is used and stored [55]:

- **Run-of-river:** This type of HPP harnesses the natural flow of a river to generate electricity. It can be built either adjacent to the river, redirecting a portion of its water through a penstock, or directly within the river utilizing a small dam. The electricity generation is directly dependent on the river's flow, which may be subject to considerable seasonal variation.
- **Storage regulation:** This type of power plant is designed to make use of a large water reservoir, sometimes even multiple ones. Water is stored during periods of high inflow and released when the flow decreases, which allows the plant to operate mostly independently of seasonal changes. The reservoir is usually located at a higher altitude than the plant itself, so the water's potential energy can be converted into both pressure and kinetic energy, enabling the plant to generate significantly more power compared to other types of plants.
- **Storage pumping:** This type of power plant takes advantage of the Francis turbine's ability to function as both a turbine and a pump. It works as a storage system, pumping water back into the reservoir when energy prices are low and releasing it later to generate power when prices are higher.
- **Tidal:** It exploits the daily tides to move water through the turbines. Its energy production is unsteady and fluctuates several times throughout the day.

2.3.1 HOW IT WORKS

Regardless of the specific type, a HPP converts the potential energy of water stored at higher elevation into electricity by channeling the flow through a turbine that drives an electric generator.

The equation that defines the energy produced by the unit was presented in [55] and is here rewritten in its most commonly used form.

$$P = \frac{\eta \rho g Q H}{1000} \quad (2.1)$$

Where:

- P is the produced power (KW)
- η is the unit efficiency (unitless)
- ρ is the density of the liquid (Kg/m^3)
- g is the gravitational acceleration (m/s^2)
- Q is the flow rate through the turbine, meaning the volume of water that passes through it per unit of time (m^3/s)
- H is the net head at the turbine (m), defined as the difference in altitude between the upper and lower basins (the head) accounting for the energy losses in the penstock.

2.3.2 OPERATING POINT

The behavior of the machine at any given time is called its Operating State (OS). When the unit is synchronized and delivering power to the grid, the set of parameters in Equation 2.1 defines the operating point.

Some parameters in this equation can be treated as constants. The gravitational constant g is fixed at the site, and the water density ρ remains essentially constant despite minor variations due to sediment. The rotation speed is also effectively constant for the purposes of this work, since the analyzed unit is a synchronous generator whose speed is fixed by the grid frequency and the number of poles whenever it is connected through the main circuit breaker.

The remaining parameters are constrained by turbine and hydraulic characteristics. The net head H depends on the flow Q because energy losses increase with the square of the water velocity v . The analyzed unit is equipped with a Francis turbine, where the flow Q is regulated by the distributor, or wicket gates, and the efficiency η depends on both Q and H .

For these reasons, the operating point in this work is described using only three variables: rotation speed (which varies only during transient OSs such as run up or coast down), power, and wicket gate opening.

2.3. HYDROELECTRIC POWER PLANT

2.3.3 SIGNALS FROM THE PLANT

Monitoring the condition of a hydroelectric unit relies on analyzing different operational signals. While each facility may track its own specific set of measurements, **vibration signals**, **temperature**, and **pressure** are almost always included. These signals provide essential information about the unit's performance and help identify abnormal operating conditions.

Among them, **vibration signals** (oscillatory motions around an equilibrium point [56]) are especially important for rotating machinery. High vibration levels can accelerate bearing wear, shorten equipment lifespan, increase noise and safety hazards, and lead to higher energy consumption. In severe cases, uncontrolled vibrations may cause major mechanical failures, resulting in costly downtime and production losses [57]. At the same time, vibration data serve as valuable indicators of issues such as imbalance, misalignment, wear, and cavitation, making them one of the most powerful tools for condition monitoring.

In the following sections, we give a brief overview of the most commonly used techniques for analyzing vibration signals, generating descriptors that are also commonly repurposed as engineered features for machine learning models.

TIME DOMAIN ANALYSIS

Time-domain analysis is the most basic way to examine plant signals. In vibration analysis, the features extracted depend on the type of signal.

The key aspect of a *relative* vibration signal is its range, since this ensures that rotating parts do not come into contact with stationary ones. A useful feature here is the **peak-to-peak** value, which measures the maximum range of the signal within a time window.

In *absolute* vibration signals, instead, the focus is on the amount of energy in the signal, as this reflects the stresses experienced by the system. The most suitable feature in this case is the RMS.

Another useful feature is the **crest factor**, defined as the ratio between the maximum amplitude (zero-peak) and the RMS within the same window. It is used to detect increases in the high-frequency components of vibration compared to the overall broadband vibration. It is particularly useful for variable-speed machinery, since both peak-to-peak and RMS naturally increase with speed [58].

FREQUENCY DOMAIN ANALYSIS

Frequency-domain, or spectral, analysis is used to break down a vibration signal into its harmonic components, making it easier to understand which phenomena contribute to

the overall vibration. By combining theory with operator expertise, these components can be linked to specific machine behaviors, each of which may indicate the development of a potential fault.

In rotor dynamics, frequency components are usually expressed in terms of harmonics. The shaft's rotation frequency is considered the fundamental frequency, also called the first-order harmonic. Other frequencies are then expressed relative to it. This representation is particularly useful when the rotation speed is not constant, such as during run-up or coast-down (when the unit starts or stops).

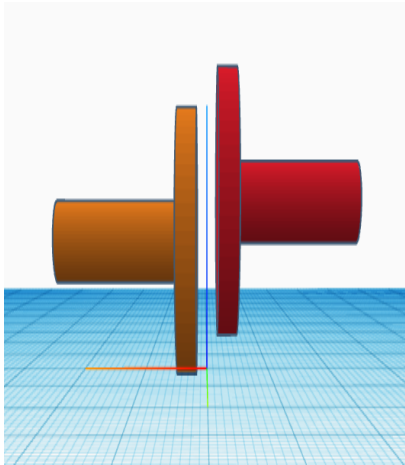
One of the most common faults detectable in the power spectrum is rotor unbalance, caused by a center of mass not aligned with the shaft axis. This may result from uneven material distribution or from radial misalignment between shaft components. Literature [59] links this condition to a peak at the synchronous frequency of the spectrum, which corresponds to the shaft's rotation speed.

Misalignment is another frequent issue. The shaft of a hydroelectric unit is a massive element, typically assembled from several segments that are flanged and bolted together. Even with careful installation methods, small construction or assembly imperfections are inevitable, leading to two types of misalignment:

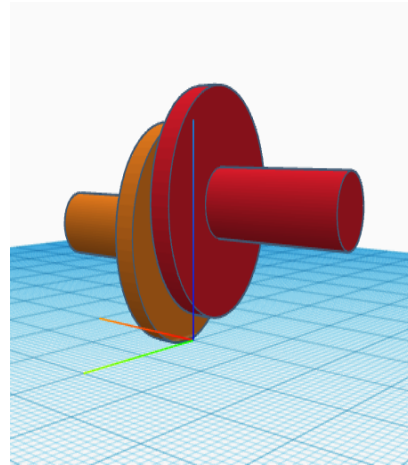
- **Radial misalignment** (or parallel offset) occurs when the axes of two components are offset, as represented in Figure 2.5. It appears in the power spectrum as an increase in the synchronous component (first-order harmonic), with the magnitude depending on the misalignment severity [60].
- **Angular misalignment** occurs when the axes of two components form an angle as represented in Figure 2.6. It can increase either the synchronous component or the second-order harmonic, depending on the type and degree of misalignment [60].

A further phenomenon specific to Francis turbines is the **vortex rope**. This is a cavitation effect that forms as a vortex near the runner outlet. It induces strong pressure pulsations, axial and radial forces, torque fluctuations, and vibrations of the turbine structure [61]. According to the literature [62], the vortex rope appears most prominently under **part-load conditions**, which typically correspond to a discharge Q between 45% and 60% of the best efficiency point (Q_{BEP}). The main harmonic of the vortex rope is usually found at 20-25% of the fundamental frequency, with additional harmonics appearing around 50% of the fundamental.

2.3. HYDROELECTRIC POWER PLANT

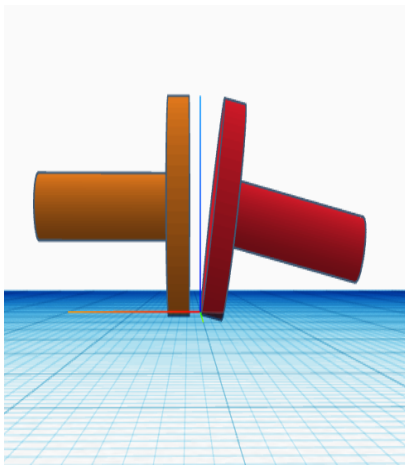


(a) Side view

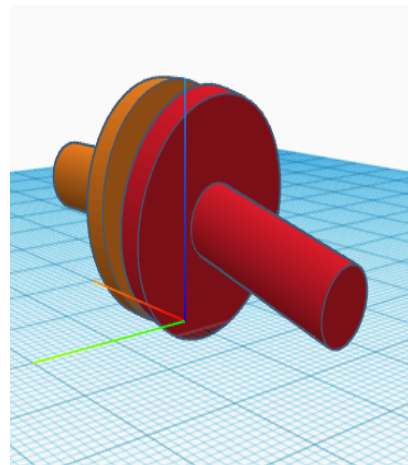


(b) Back view

Figure 2.5: Example of parallel misalignment of the shaft components



(a) Side view



(b) Back view

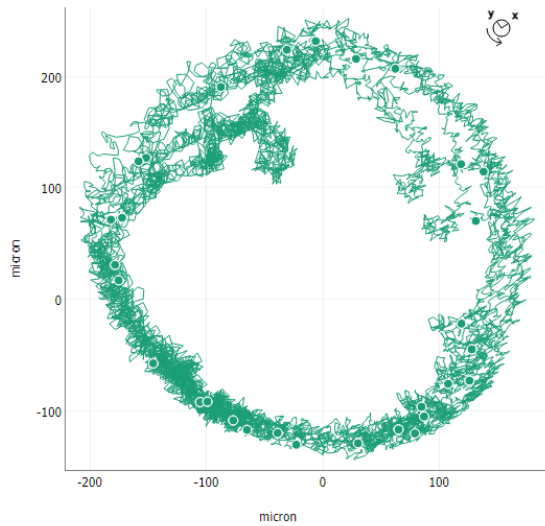
Figure 2.6: Example of angular misalignment of the shaft components

ORBIT ANALYSIS

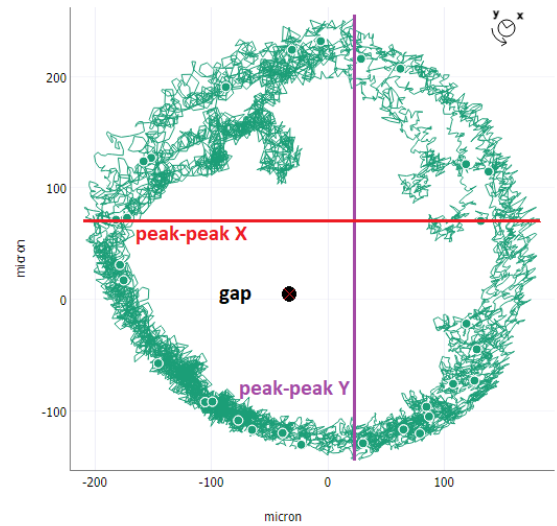
The orbit traced by the shaft inside the bearings, shown in Figure 2.7a, is a key parameter to monitor. The **constructive gap** (the clearance inside the bearing) sets the maximum safe limit of shaft motion, beyond which the system risks serious damage.

To measure the orbit, two sensors are installed on orthogonal axes to capture the relative displacement of the shaft with respect to the bearing pads. The bearings operate with an oil film that, during rotation, keeps the shaft only a few hundredths of a millimeter away from the closest pad in the load direction. From these signals, the most common features extracted

are: the **peak-to-peak** values of both sensors, the relative **phase shift** between them, and the **gap**, which represents the center of overall shaft rotation (Figure 2.7b).



(a) Orbit reconstructed from shaft X and Y relative displacements.



(b) Time domain features extracted from the orbit raw signal.

Figure 2.7: Orbit plot and relative time domain features (images courtesy of [63]).

2.4 MACHINE LEARNING

The idea of creating machines that can learn and improve from experience has been present since the earliest days of computing. In 1950, Alan Turing anticipated this vision in his seminal paper [64], laying the foundation for what would later become Machine Learning (ML). Over the following decades, research from diverse fields such as statistics, AI, information theory, biology, and control theory converged into the discipline of ML [65].

The motivation for ML lies in the existence of problems where explicit programming is impractical or impossible [66]. Algorithms capable of learning from data are essential in three main scenarios:

- When the **knowledge of the problem** is incomplete or too complex to encode manually. A typical example is natural language processing, where linguistic rules and vocabulary evolve continuously [67].

2.4. MACHINE LEARNING

- When the **environment** cannot be fully modeled or is subject to change over time. In real-world deployments, conditions often deviate from controlled assumptions, degrading algorithmic performance.
- When there is a need to **discover hidden structure** in data, uncovering patterns not evident to human designers.

A widely cited definition by Mitchell [65] formalizes ML as follows: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ."*

2.4.1 SUPERVISED LEARNING

What defines supervised learning is the availability of an outcome variable that guides the learning process [68]. Each training input, usually denoted as $x_i \in X$, is associated with a label $y_i \in Y$. The objective is to learn a function $f : X \rightarrow Y$ that not only fits the training data but also generalizes well to unseen inputs [23].

A key strength of supervised learning is that the presence of labels makes it straightforward to define performance metrics, often referred to as cost functions, loss functions, or error functions. This allows the training process to be framed as an optimization problem, where the goal is to minimize the discrepancy between predicted and true labels [69].

Supervised learning is commonly divided into two main tasks depending on the type of label:

Classification: where the labels are categorical and belong to a finite set. Formally, the task can be defined as learning a mapping $f : \mathbb{R}^n \rightarrow \{1, 2, \dots, K\}$, where K is the number of classes and n is the dimensionality of the input space [69]. This category can be further specialized into *binary classification* ($K = 2$), *multi-class classification* ($K > 2$), and *multi-label classification*, where each instance can simultaneously belong to multiple classes, i.e. $y_i \subseteq Y$ [70]. A classical example is handwritten digit recognition [71].

Regression: where the labels are continuous values, typically real numbers. The task is commonly defined as approximating an unknown function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ [69], with the natural extension $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ [23]. A relevant example in the context of PdM is RUL, which is further discussed in Section 2.2.

Supervised models are widely applied in industry for both classification and regression tasks, offering reliable solutions across many domains [72]. Some of the most commonly

used algorithms include the following:

- Support Vector Machine (SVM) [73] has seen broad adoption in industrial applications such as fault detection in manufacturing [74] and quality control [75]. Its main advantages are the ability to separate data with clear margins and to handle high-dimensional data. On the downside, it is often slower to train and run than more modern approaches, and it lacks of interpretability [76].
- K-Nearest Neighbors (KNN) [77] is applied in areas such as PdM [78], process monitoring [79], and energy load forecasting [80]. Its appeal lies in its simplicity and effectiveness in capturing local decision boundaries.
- Tree-based ensemble methods, including Random Forest (RF) [81] and Extreme Gradient Boosting (XGBoost) [82], are also widely used in industry. RF in particular has proven effective in tasks such as fault detection [83] and quality control [84]. Its strength comes from combining multiple decision trees, which helps reduce overfitting while improving accuracy. As reported in [85], compared to other ML approaches, RF proves relatively robust even when the number of features exceeds that of observations, thanks to their random feature selection and ensemble averaging. Another advantage is that they are naturally more interpretable than many other ML methods, since decision trees, linear models and rules remain among the most transparent models available [86].

2.4.2 UNSUPERVISED LEARNING

Unsupervised learning aims at identifying patterns, structures, or other informative properties directly from unlabeled data [69]. Although measuring performance is less straightforward than in supervised learning, common approaches exist. These can be divided into *extrinsic evaluation*, which uses external information such as labels, and *intrinsic evaluation*, which relies solely on the data and the model's internal properties [87]. While intrinsic metrics can provide useful guidance, they do not always reflect what users consider meaningful or relevant to the task.

Unsupervised learning encompasses a broader set of tasks than supervised learning. The following sections provide an overview of the most common approaches.

Clustering. Clustering is a central unsupervised learning technique that consists of grouping data into subsets based on similarity or distance measures. It uncovers patterns and

2.4. MACHINE LEARNING

structures in the data without requiring prior knowledge of group labels. The objective is to ensure that points within the same cluster are more similar to each other than to points in different clusters [88]. Widely used clustering algorithms include K-means [89], Mean Shift [90], and DBSCAN [91]. In industrial settings, clustering supports tasks ranging from exploratory data analysis and the identification of operational modes to serving as a preliminary step for detecting unusual behaviors before deploying more specialized models [92, 93].

Dimensionality Reduction. Dimensionality reduction techniques simplify datasets by reducing the number of variables while retaining essential structure and information [94]. Principal Component Analysis (PCA) [95] is the most widely used linear approach, while nonlinear methods such as t-Distributed Stochastic Neighbor Embedding [96] and Uniform Manifold Approximation and Projection [97] are particularly effective for analyzing complex, high-dimensional data. Autoencoders can also be used, which are NN-based methods trained to learn an invertible mapping between the original dataset and a lower-dimensional representation. They consist of two parts: an encoder that compresses the input into a compact latent space, and a decoder that reconstructs the original data from this representation [98]. The training objective is to minimize the reconstruction error, typically expressed as the Mean Squared Error (MSE) between the input and its reconstruction. A more detailed discussion of Autoencoders will be provided in Chapter 4. In industrial applications, where datasets often contain many features, dimensionality reduction is commonly used as a pre-processing step to highlight the most informative aspects of sensor data [99]. This not only streamlines analysis but also improves model efficiency [72], and it can also serve as a tool for visualizing high-dimensional structures [100].

Density Estimation. Density estimation is an unsupervised task focused on estimating the underlying probability distribution from which a dataset is drawn [101]. The primary goal is to approximate the data's probability density function or probability mass function. Common approaches include parametric models, such as Gaussian Mixture Models [102], and non-parametric techniques like Kernel Density Estimation (KDE) [103]. In practical applications, density estimation is often critical for pinpointing outliers that significantly diverge from the learned distribution [104], generating synthetic datasets that accurately reflect the original data [105], or gaining a deeper understanding of the data's intrinsic patterns [101].

Association Rule Learning. Association rule learning is an unsupervised method designed to uncover significant relationships or associations between items within large datasets. The main objective is to extract rules that capture strong attribute correlations, typically

expressed in the form "if A, then B" [106]. The strength and relevance of these rules are commonly assessed using metrics such as support, confidence, and lift [107]. The Apriori algorithm [108] is one of the most established techniques, used to identify frequent itemsets and generate rules from them. Association rule learning is widely applied in market basket analysis to uncover customer purchasing patterns [109]. Beyond retail, it has proven useful in domains such as medical diagnosis, where it helps identify common symptom combinations [110], and in cybersecurity, where it aids in detecting suspicious network behaviors [111].

2.4.3 SEMI-SUPERVISED LEARNING

Semi-supervised learning is a method that combines labeled and unlabeled data so that each supports the other in improving model performance [112]. This approach is particularly useful in industrial applications, where labeled data are scarce and costly to obtain, while large amounts of unlabeled data are readily available. As a result, it is also a central pillar of this work, as in Chapter 7 it allows us to balance between the complete information obtained with a supervised approach and the inherent robustness and adaptability of unsupervised methods.

A necessary condition for semi-supervised learning is that the underlying marginal distribution of the input data, $p(x)$, contains information about the posterior distribution $p(y|x)$. If this condition holds, unlabeled data can provide useful information about $p(x)$, which in turn can improve estimates of $p(y|x)$. If the condition is not met, however, the unlabeled data offer no benefit and cannot improve predictive accuracy [112, 113].

This learning paradigm will be further explored in Chapter 7. The discussion is intentionally placed there to keep it close to the proposed method, ensuring a complete understanding of its implementation and mechanics.

2.4.4 REINFORCEMENT LEARNING

In Reinforcement Learning (RL) an agent learns to make decisions by interacting with an environment and observing the results of its actions with the goal to maximize the cumulative reward over time. Unlike supervised learning, where models are trained on labeled datasets, RL relies on trial-and-error: the agent tries actions, sees the outcomes, and gradually improves its strategy. This process allows the agent to discover effective ways to achieve its goals.

Key concepts in RL include:

- Agent: The learner or decision-maker that interacts with the environment according to

2.4. MACHINE LEARNING

its policy.

- **Environment:** The external system providing feedback in the form of rewards or penalties.
- **State:** A representation of the current situation or configuration of the environment.
- **Action:** A decision taken by the agent that influences the state of the environment.
- **Reward:** A scalar signal received after taking an action, indicating the immediate benefit. Rewards can be sparse or delayed, which adds complexity to the learning process.

While RL is not the primary focus of this work, it has found numerous valuable applications in industrial settings. To provide context, it is useful to briefly trace the key milestones in its development. The multi-armed bandit problem was first formulated in 1952 [114], laying the groundwork for sequential decision-making. This was followed by the introduction of Temporal Difference Learning [115], which combined ideas from dynamic programming and trial-and-error learning. Subsequent advances include Q-Learning [116], which allowed agents to learn optimal action policies without a model of the environment, Deep Q-Networks [117], which extended Q-Learning to high-dimensional inputs using deep neural networks, and more recent algorithms such as Proximal Policy Optimization [118], which improved stability and efficiency in policy gradient methods.

RL is extensively applied across various industrial sectors, including energy and manufacturing. Within HPP, RL is primarily used to optimize reservoir management and operations [119]. In robotics, RL allows machines to learn intricate skills, such as high-precision assembly, through direct environmental interaction [120]. PdM presents another promising area, where RL assists in early fault detection and in calculating health indicators to guide maintenance strategies [121]. Furthermore, RL has been utilized for process control [122] and for enhancing routing and inventory decisions in supply chain management [123].

2.4.5 DEEP LEARNING

Deep Learning (DL) is a branch of ML that has gained significant attention in industry for its ability to automatically learn complex hierarchical representations from raw data, reducing the need for costly and time-consuming feature engineering [69]. This makes it especially effective for unstructured data such as images, audio, and text, which play an increasingly important role in modern industrial processes.

In practice, DL has driven progress across a wide range of industrial applications. Convolutional Neural Networks (CNNs) [124], for instance, are widely used in automated

visual inspection for manufacturing quality control and defect detection in materials [125]. RNNs [126] and more recent architectures such as transformers [127] are applied in PdM [128] and energy demand forecasting [129]. These approaches have also been adopted in industrial robotics, enabling more adaptive and autonomous decision-making [130].

The success of DL in these areas has been driven by the availability of large-scale industrial datasets, the growing accessibility of high-performance computing (particularly GPUs and TPUs). Furthermore, the introduction of training techniques to prevent overfitting, such as dropout [131] and batch normalization [132].

2.4.6 ANOMALY DETECTION

AD is a foundational concept in this thesis, playing a central role in Chapters 4, 6, and 7. According to [133], *"Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior"*.

Research on AD has a long history, with early discussions already appearing in the 19th century [134, 135]. Since then, the field has grown steadily and today AD methods are widely applied in domains as diverse as fraud detection [136], bot detection [137], and PdM in industry 4.0 [138].

Most AD techniques follow an unsupervised learning paradigm, leveraging the assumption that the vast majority of data points are normal and only a small minority represent anomalies. While this assumption often holds true, alternative approaches also exist, including semi-supervised approaches [139], and more recent self-supervised strategies [140].

The main categories of algorithms used for AD can be grouped as follows:

- **Statistical methods** were the earliest approaches, originating directly from the field of statistics. These methods model the probability distribution function of normal data and classify points with low probability as anomalies. Examples include KDE [141, 104] and the more recent Empirical Cumulative Distribution-based Outlier Detection [142].
- **Proximity-based methods** rely on the principle that anomalies are often isolated with respect to distance from other points, assuming the training set is representative. Classic examples include the Local Outlier Factor [143], as well as more recent approaches such as Rotation-based Outlier Detection [144].
- **Machine learning models** extend AD to more complex settings. Common techniques include One-Class Support Vector Machine (OCSVM) [145], and DL approaches like the already mentioned Autoencoder.

2.5. MACHINE LEARNING CHALLENGES IN PREDICTIVE MAINTENANCE

- **Ensemble methods** combine multiple models to capture different types of anomalies, coordinating their predictions into a single decision. This allows each model to specialize, improving robustness. Examples include iForest [146], Extreme Boosting Based Outlier Detection [147] and Kit-Net [148].

Applications of AD in industrial contexts are particularly prominent in PdM, where early detection of abnormal patterns in sensor data can prevent costly equipment failures and reduce downtime like pointed out in [149]. In manufacturing, AD is widely used for quality control, enabling the automatic identification of defective products or deviations in production processes [150].

2.5 MACHINE LEARNING CHALLENGES IN PREDICTIVE MAINTENANCE

The deployment of ML techniques in industrial environments faces challenges ranging from data quality to computational and operational constraints. These challenges directly influence the reliability, scalability, and practical use of predictive models in real-world scenarios. As reported in [151], these challenges can be categorized into the following groups: real-time processing and infrastructure; data quality; data distribution shift; and human-machine interaction.

2.5.1 REAL-TIME PROCESSING AND INFRASTRUCTURE

In safety-critical industrial environments, the massive volumes of data generated by sensors, logs, and operational records must be processed in real time to detect and prevent faults before they escalate. This requirement becomes particularly demanding when computationally intensive ML techniques are applied, as they place heavy pressure on infrastructure for data storage, processing, and analysis.

While cloud computing offers scalable resources, its effectiveness is often limited by bandwidth and latency constraints, while the transmission of sensitive data can raise privacy and security concerns. Edge computing, instead, is increasingly adopted to bring computation closer to the data source, reducing delays and ensuring safer control over the data. However, the success of edge solutions depends on the use of lightweight models that can operate **efficiently on resource-constrained devices**. These models are specifically designed to balance accuracy with computational efficiency, making real-time industrial data analysis both feasible and practical in environments where reliability and speed are critical.

2.5.2 DATA QUALITY

In industrial scenarios, data often suffers from issues that can undermine the performance of ML models. The simplest of these is **noise**. While excessive noise can render data unusable, most ML models are fairly robust to it, and in some cases, noise can even act as a form of regularization during training, improving generalization [152].

A more challenging problem arises from the scarcity of faults, which are rare events. Accordingly, datasets are primarily composed of data representing normal operation, with only limited examples of faults. Using supervised models is generally preferred when the data quantity and quality allow it because of their superior performance and the richer information they provide. However, in industrial contexts this is rarely the case. The data available for gathering often results in **imbalanced datasets**, which, with supervised approaches, can lead to biased predictions and poor generalization. Strategies to address this issue include over-sampling minority classes, undersampling majority classes, using semi-supervised methods, or adopting unsupervised AD approaches.

Closely related to imbalance is the **labeled data scarcity**. In industrial contexts, labeling requires manual inspections and classification by domain experts. Moreover, since faults occur infrequently, it is also difficult to collect a complete and representative dataset for every possible fault scenario, leaving gaps in model coverage.

Manual labeling, combined with the continuous and high-dimensional nature of industrial data and the complexity of fault conditions, can also introduce **inconsistencies in the labels**. These errors often appear in ambiguous cases, such as when fault boundaries are unclear or when multiple faults occur simultaneously, making it difficult to assign a single correct label. Still, even small labeling inaccuracies can be particularly harmful when labeled data is limited, as they may lead the model to learn incorrect associations. This issue is discussed in detail in Chapter 7.

Finally, industrial data frequently contains **missing values**, caused by sensor failures, communication problems, or data corruption. Missing entries reduce dataset completeness and negatively affect model training and performance. Addressing missing values typically requires imputation techniques or algorithms designed to handle incomplete data, both of which increase the complexity of the development process.

2.5.3 DATA DISTRIBUTION SHIFT

In dynamic industrial environments such as HPPs, which are subject to varying operational conditions, the statistical properties of data can change over time. This leads to a

2.5. MACHINE LEARNING CHALLENGES IN PREDICTIVE MAINTENANCE

phenomenon known as data distribution shift. Data distribution shift is defined in [153] as the phenomenon in which the generated data changes over time, causing the model's predictions to become less accurate.

The cause of this disruptive effect can be traced back to the fact that most ML models operate under the assumption that training and test data are drawn from the same distribution. When this assumption is violated, models may struggle to generalize to new data.

The authors of [153] categorize data distribution shifts into five main types. Defining X as the input of the model and Y as the set of possible labels, these types are:

- **Covariate Shift:** which occurs when the distribution of input data $P(X)$ changes, while the relationship between inputs and outputs $P(Y|X)$ remains constant. An example in a manufacturing scenario could consist in changes in operational practices making some system states more common than others, but faulty patterns remain faulty.
- **Label Shift:** which occurs when the distribution of labels $P(Y)$ changes, while the conditional distribution of inputs given labels $P(X|Y)$ stays the same. An example of this is a change in material quality leading to an increase in certain types of faults, while the characteristics of each fault type remain unchanged.
- **Concept Drift:** which occurs when the relationship between inputs and outputs $P(Y|X)$ changes over time. As machinery ages or undergoes maintenance, the patterns that indicate faults may evolve, requiring models to adapt to these new relationships.
- **Feature Change:** which occurs when the features used by the model change, either due to the addition of new sensors, removal of existing ones, or changes in data collection or preprocessing methods.
- **Label Schema Change:** which occurs when the definitions or categories of labels change over time. In classification tasks this happens when new fault types are introduced or existing ones are redefined, while in regression tasks this happens when the target variable is redefined or measured differently, such as a change in range.

There is no one-size-fits-all solution to address all data distribution shifts. However, in the literature we can find several strategies to mitigate the impact of selected types of shifts. Depending on the specific context, a combination of these strategies can be employed to maintain model performance over time. Some of these strategies include:

- **Massive Training:** As reported in [153], this is the SOTA approach in the field. If the training dataset is large enough, the model will learn such a comprehensive distri-

bution that most data points encountered in deployment will likely come from this distribution.

- **Retraining and Updating Models:** Periodically retraining models with recent data can help them adapt to changes in data distribution. This strategy is commonly used in industrial scenarios, where retraining is scheduled on a regular basis or triggered by performance degradation.
- **Active Learning**[154]: Involving human experts in the loop to label new or uncertain data points can help improve model performance in the face of distribution shifts. This is particularly useful when labeled data is scarce or when new fault types emerge.
- **Online Learning**[155]: This approach allows models to update their parameters incrementally as new data arrives, enabling them to adapt continuously to changing distributions.
- **Incremental Learning, Continual Learning, and Lifelong Learning**[13]: These techniques are specifically designed to allow models to learn from new data without forgetting previously acquired knowledge. This approach is a valid alternative to the previous ones due to reduced data and computational requirements. The terminology is not always consistent and often changes over time.

2.5.4 HUMAN-MACHINE INTERACTION

As introduced in Chapter 1, human-machine interaction plays a key role in industrial environments and is central to the transition towards Industry 5.0. In cyber-physical production systems, workers are the most flexible element, taking on a wide variety of tasks ranging from specification and monitoring to modifying production strategies [156]. Bringing ML models into these workflows makes the picture more complex, since it concerns not only the algorithms themselves but also how people interact with them. This means designing interfaces that are intuitive to use, making sure predictions are explained in a way that operators can actually understand and trust, and creating conditions where humans can step in and take control whenever needed. In this way, responsibility and oversight remain with the human operators, while still benefiting from the efficiency of automation.

In practice, two main challenges stand out. The first is helping the human operator to **trust and understand** the model's predictions. This becomes especially important in safety-critical settings, where a wrong choice could have serious consequences. Work in this area is usually referred to as eXplainable Artificial Intelligence (XAI), with a comprehensive overview

2.6. SUMMARY

provided in [157]. A variety of approaches exist, ranging from models that are inherently interpretable to post-hoc explanation methods, which can be either model-agnostic or model-specific. Explanations can also be categorized as *global*, giving insight into the model's overall behavior, or *local*, focusing on individual predictions. Local explanations will be part of the approaches presented in Chapters 4 and 6, while global explanations are used in Chapter 6.

The second challenge is making sure operators can **intervene and correct** the model when necessary. Addressing this requires systems to provide straightforward ways for humans to give feedback or corrections, so that they can guide the model's learning process and help it improve over time. According to [158], there are three main levels of human involvement in the ML process:

- **Active Learning** [154], where the system leads the learning process and uses humans mainly as oracles to label unlabeled data.
- **Interactive Machine Learning** [159], where humans and systems work together more closely, with people providing feedback in a more targeted, frequent, and incremental way than in traditional ML.
- **Machine Teaching** [160], where domain experts take charge of the learning process by deciding what knowledge should be transferred to the model.

2.6 SUMMARY

In this chapter, we introduced the industrial context relevant to this thesis, with a focus on data-driven PdM and the challenges that ML solutions face in this domain.

In summary, while many ML applications have become de facto standards for tasks across the industrial domain, their direct adoption is often hindered by challenges that differ across tasks and contexts. The goal of this thesis is to present a toolbox to address some of these challenges in the setting of PdM for HPPs and that can be adapted to diverse contexts with minimal modifications. This toolbox is designed to allow users to select the most suitable methods for their specific needs.



Research Framework

The core motivation of this thesis is to enhance PdM for HPPs. Hydroelectric facilities are critical assets in ensuring energy security [8, 9] and supporting the transition toward renewable energy sources. At the same time, their engineering complexity and the strict safety requirements that govern their operation make maintenance both essential and challenging. As presented in Chapter 2, in recent years data-driven approaches such as ML and DL have been increasingly adopted across a wide range of application domains. The models proposed in the literature are typically designed and evaluated as general-purpose tools, as appropriate for methodological research. However, the effects of practical challenges that are common in industrial scenarios, and that become even more critical in the context of HPP, are rarely taken into account. These challenges include the limited availability of labeled fault data, contamination within nominal operational data, and the continual evolution of sensor infrastructures. In addition, in line with Industry 5.0, human-centric design is becoming increasingly important in industrial contexts. Algorithms should not only produce outputs that are interpretable by humans but also incorporate their directives. This bidirectional interaction builds trust in automated systems and enables effective collaboration between machine intelligence and human expertise, enhancing safety and reliability. In the remainder of this chapter, we first present the three main research directions and explain the role of proxy tasks (Section 3.1). We then introduce the general problem definition and assumptions guiding our experiments (Section 3.2). Finally, we discuss how the presented methods connect and how their combined contributions advance PdM in hydroelectric systems (Section 3.3).

3.1 RESEARCH DIRECTIONS AND PROXY TASKS

The methodological contributions of this thesis are organized around three main research directions for improving PdM: *efficiency*, *robustness*, and *human-in-the-loop*. Each direction reflects concrete challenges faced in industrial contexts and, particularly, in hydroelectric systems.

- **Efficiency.** The goal is to allow models to adapt to dynamic environments and tasks while reducing costs in terms of computational resources, processing time, and memory footprint. This is crucial in hydroelectric systems where sensor infrastructures evolve continuously, requiring methods that can integrate new data without full retraining.
- **Robustness.** Industrial scenarios rarely provide ideal conditions. Data collection and labeling are extremely expensive, measurements are often noisy, and positive fault instances are rare, and difficult to obtain without simulation. Robust models must therefore deliver reliable results with scarce, imperfect, and contaminated data.
- **Human-in-the-loop.** In high-risk environments such as HPP, interpretability of model outputs is essential for trust and safety [15]. At the same time, enabling operators to inject their knowledge directly into the model helps reduce semantic gaps. As defined in [161], the semantic gap is *the lack of coincidence between the information that one can extract from the visual data and the interpretation that the same data have for a user in a given situation*. Closing this gap improves collaboration between operators and models, fostering safety, reliability and confidence in automated systems.

A key difficulty in developing methods for hydroelectric PdM is the lack of standardized and openly available datasets. HPPs generate complex and heterogeneous data, but companies rarely share it due to safety concerns, competitive advantage, and privacy reasons. The vast majority of publicly available datasets contain aggregated information such as energy production, power plant location, reservoir water levels, or weather data (e.g., [162]), which are not directly relevant for PdM. The few datasets that focus on specific components (e.g., bearing vibration data [163]) capture only narrow aspects of the overall system complexity and are thus rarely used in practice.

The first project (Section 4) highlights both the advantages and drawbacks of using proprietary data. On the one hand, we had access to a diversified collection of faults and were able to exploit the high specificity of the features together with the extensive knowledge of domain experts to effectively perform interpretability guided RCA. On the other hand, a

complete point-wise labeling of the data was not feasible, and we could only rely on anomalous segments of data as a reference. Moreover, the inability to share the dataset reduces replicability, extensions and improvements from the research community. We considered this tradeoff acceptable, as the work adapts established tools to a new context, and future validation or extensions can still be pursued on alternative datasets.

The other three projects presented in this thesis rely on proxy tasks from other domains such as transportation and industrial fault detection. Each proxy task is paired with a clear hydroelectric formulation that illustrates how the proposed method would operate in practice. This dual approach enables validation on widely known datasets, ensuring comparability with the broader research community, while preserving a consistent focus on hydroelectric PdM.

3.1.1 MAPPING PROJECTS TO RESEARCH DIRECTIONS

The four projects can be mapped to the three research directions as follows (see Figure 3.1):

- **Anomaly Detection and Human-in-the-Loop for Root Cause Analysis [18]:** Section 4 shows how AD and interpretability methods can be combined into a *human-in-the-loop* support tool for operators performing RCA. The study relies on a proprietary hydroelectric dataset.
- **Continual Learning for Behavior-Based Driver Identification [19]:** Section 5 addresses *efficiency* through CL, enabling models to recognize new drivers without full retraining. This approach supports in-vehicle adaptation with low computational and memory costs. Experiments are carried out on the OCSLab dataset [164].
- **Feature-Evolving Anomaly Detection:** Section 6 explores *efficiency* under feature-evolving scenarios, where new sensor inputs are added over time. The proposed RootIF adapts to new features while preserving previously learned knowledge, without requiring storage of historical data. Experiments use of open collection of datasets from ODDS [165].
- **Robust Fault Detection and Identification:** FLEX-C is a *robust* fault detection and identification framework designed for scarce and contaminated data, evaluated using the open datasets Statlog Satellite [166] and Glass Identification [167]. It generalizes the ideas of Active Learning Isolation Forest (ALIF) [168] and Bayesian Active Learning Isolation Forest (BALIF) [169] from anomaly detection to classification. Although the extensions toward *efficiency* and *human-in-the-loop* still require validation in future work

3.1. RESEARCH DIRECTIONS AND PROXY TASKS

(see Section 7), the ensemble backbone based on iForest makes it possible to integrate ALIF and BALIF to adapt the model to the user's definition of *normality*, while the simplicity of the weak classifiers ensures that incremental updates remain feasible, allowing new user-defined labels to be incorporated without retraining the entire model.

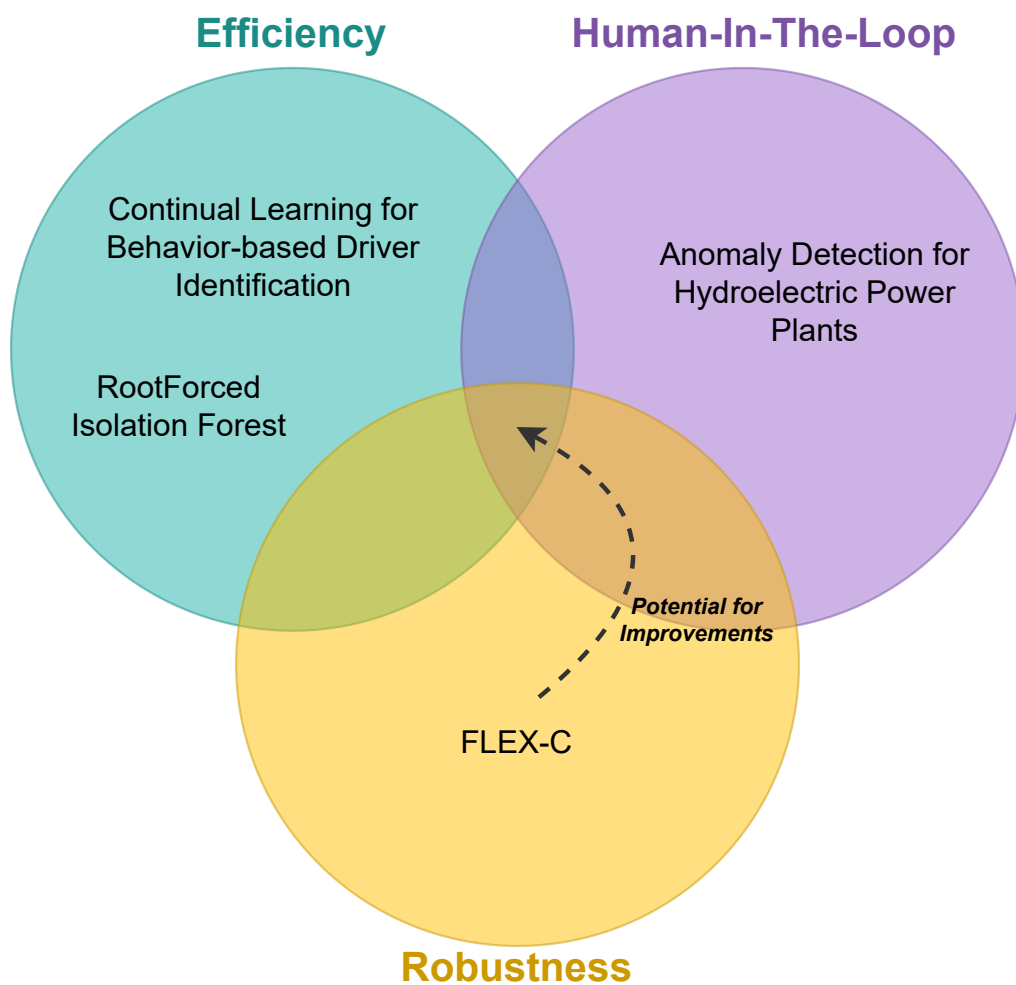


Figure 3.1: Mapping of the four projects to the three research directions: efficiency, robustness, and human-in-the-loop.

3.2 GENERAL PROBLEM DEFINITION

Data from HPP systems, and industrial systems in general, is produced as a continuous stream from sensors monitoring the equipment. From this stream, engineered features can be computed. If a storage infrastructure is available, both raw signals and extracted features are archived as tabular data, with each record linked to its timestamp.

We define the problem in general terms to cover all scenarios addressed in this work. The data stream and its processing pipeline can be represented in an F -dimensional real space, where F is the number of features. Formally, the data is a sequence $[x_0, \dots, x_n]$ with $x_i \in \mathbb{R}^F, \forall i$.

In FeV scenarios, the number of features can increase over time, leading to a sequence $F_0 \leq \dots \leq F_n$ with $x_i \in \mathbb{R}^{F_n}, \forall i$. In such cases, features unavailable in earlier stages are treated as hidden but are assumed to exist since they become measurable once the corresponding sensors become available.

Each data instance x_i is associated with a true label $y_i \in \mathcal{C}$, where $\mathcal{C}$ is the set of possible fault types including the *normal* class. In Fault Detection tasks, $\mathcal{C}$ reduces to $\{\text{normal}, \text{fault}\}$. If the number of classes increases over time, the true class set $\mathcal{C}$ remains fixed, since it represents the hypothetical correct set of fault types for the system. What changes is our ability to label instances for the training set. This can be modeled as a sequence of increasingly complete labeling sets $\mathcal{C}'_0 \subseteq \dots \subseteq \mathcal{C}'_n = \mathcal{C}$, where $\mathcal{C}'_k$ contains the classes available for labeling at time step k .

We assume the training data to be a collection of i.i.d. instances in one of the following settings:

- **Unsupervised:** only unlabeled data $[x_0, \dots, x_n]$ is available.
- **Supervised:** each instance has a label (not necessarily the true one), yielding pairs $[(x_0, y'_0), \dots, (x_n, y'_n)]$.
- **Semi-supervised:** two training sets are provided, one with unlabeled data and one with labeled data.

This framework applies across all projects, despite their specific peculiarities. In each project, the problem may be reformulated in an equivalent way to simplify the exposition. In CL, for example, we use windows of signals as training instances. While points within a window are not i.i.d., if we concatenate the features within the window, the resulting instances can be treated as i.i.d. samples, since randomly drawn windows are independent.

3.3. SYNERGIES

Given a family of models parameterized by learnable parameters w , the goal is to find the model $\theta(w^*)$ such that

$$w^* = \arg \min_w \sum_i \mathcal{L}(\theta_w(x_i), y_i), \quad (3.1)$$

where y_i is the true label associated with x_i and $\mathcal{L}$ is a suitable loss function measuring the discrepancy between the model output and the true label.

3.3 SYNERGIES

Among the four projects, FLEX-C occupies a central role in the overall framework due to its general and modular formulation. Its design allows it to interact conceptually with the other methods, highlighting potential integrations and complementary uses.

1. **Anomaly Detection and Human-in-the-Loop for Root Cause Analysis:** This project is the most general, as SHAP is a model-agnostic interpretability algorithm. It can be applied to the output of any model, making it possible to link it to the results of the other projects, providing interpretability and operator guidance, whether for FeV with RootIF or to understand the outputs of FLEX-C.
2. **Continual Learning strategies:** In general, CL methods are often designed for specific model families and do not always integrate easily with AD. An example is Elastic Weight Consolidation (EWC) being designed for parametric models trained via gradient descent. This affects their compatibility with FLEX-C depending on the weak classifiers used. The synergy is limited for the classifiers we selected, but other types of weak classifiers could enable better integration.
3. **RootIF:** This method extends iForest, making it a valid replacement for the FLEX-C ensemble structure. Previously trained classifiers operate only on the features they can process, while new classifiers can be trained on the leaves of the newly added trees. This allows incremental expansion of the feature space without disrupting the previously learned knowledge.

4

Anomaly Detection and Human-in-the-Loop for Root Cause Analysis

In this chapter, we present our approach to fault detection and RCA in the context of vibration monitoring for HPPs, based on our earlier work in [18]. In safety-critical settings like this, human experts must always remain responsible for decisions regarding operations and maintenance. However, the complexity of these systems and the sheer amount of data available can easily overwhelm them. This often results in wasted time analyzing normal situations, diverting attention from real anomalies, increasing costs, and reducing safety.

Consequently, relying on automated decisions, as in a supervised learning approach, is not practical. Data quality issues, which we will detail later in this chapter, make such methods difficult to apply. More importantly, any automated decision-making system would be redundant since the human expert must always retain final authority. Our approach instead combines a carefully selected AD model with an interpretability tool, SHAP, to support experts in their decision-making. The AD model highlights the most anomalous cases, helping experts focus their attention more effectively and reducing the risk of missed faults. The interpretability module explains which features most influenced the model's decision, allowing experts to leverage their domain knowledge and the physical meaning of the features to perform RCA.

Our approach applies established methods to real data from a HPP with the aim of delivering a solution based on industry ready technologies that can be adopted with minimal effort. The main challenge is to provide users with high quality information despite the

4.1. BACKGROUND AND RELATED WORK

limitations in data quantity and data quality that characterize the hydroelectric PdM context. The approach builds on the feature extraction procedures already in use within the company, which embed extensive domain knowledge about the relationships between these features and potential faults. This choice facilitates the effective use of expert insight and supports smoother integration of the proposed solution into existing practice.

The main contributions of this chapter are:

- We propose a methodology that combines AD and interpretability to support human experts in fault detection and RCA for vibration monitoring in HPPs.
- We evaluate several AD models on real HPPs data to identify the one best suited to assisting experts in fault detection.
- We demonstrate how SHAP-based interpretability effectively guides experts in RCA by linking engineered features with expert knowledge of faults.

The chapter is structured as follows: Section 4.1 reviews background concepts and related work on interpretability. Section 4.2 describes the dataset. Section 4.3 details our methodological choices and implementation. Section 4.5 presents the outcomes of our AD model comparison and RCA using SHAP. Finally, Section 4.6 summarizes our findings.

4.1 BACKGROUND AND RELATED WORK

In this section, we provide the necessary background to support the contributions of this work and to contextualize the subsequent chapters. Section 4.1.1 introduces the models considered in our experiments. Section 4.1.2 presents the concept of interpretability with a focus on SHAP. Finally, Section 4.1.3 reviews related work relevant to our study.

4.1.1 MODELS

In the following, we will briefly describe the models we used for the experiments. iForest will be described in more detail since it is a foundational model for this thesis and will be relevant also in Chapters 6 and 7.

ISOLATION FOREST

Isolation Forest (iForest) [146] is an unsupervised AD algorithm that operates on the principle of isolating anomalies rather than profiling normal data points. The fundamental

idea is that anomalies are "few and different," which makes them more susceptible to isolation than normal points. This algorithm builds an ensemble of decision trees, called Isolation Trees (iTrees), to partition the data. In these trees, anomalies are more likely to be separated closer to the root, resulting in shorter path lengths, while normal points are more likely to be found deeper, requiring more splits to be isolated.

Algorithm Description The algorithm consists of two main stages: training and inference.

Training: In the training phase, a forest of iTrees is constructed. To build each iTree, a random subsample of the training data is selected. The tree is then grown by recursively partitioning this subsample. At each node, a random feature is selected, and a random split value is chosen between the minimum and maximum values of that feature in the current node. This process continues until a node contains only a single instance or the tree reaches a predefined maximum depth. The randomness in feature and split value selection ensures that each iTree in the forest is unique, which contributes to the robustness of the model.

Inference: During the inference phase, each instance is passed down every trained iTree to determine the path length required to isolate it. The path length, denoted as $h(x)$, is the number of edges traversed from the root node to a terminal node for a given instance x .

Anomaly Score Computation The anomaly score for an instance is derived from the average path length of that instance across all the iTrees in the forest. To normalize this path length, it is compared to the average path length for an unsuccessful search in a Binary Search Tree (BST), which serves as a baseline for the average path length of a tree with the same number of instances. The average path length of an unsuccessful search in a BST with n instances is given by:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n} \quad (4.1)$$

where $H(k)$ is the harmonic number, which can be approximated as $\ln(k) + \gamma$, with $\gamma \approx 0.5772156649$ being the Euler-Mascheroni constant.

The anomaly score, $s(x, \psi)$, for an instance x given a subsample of size ψ is calculated as follows:

$$s(x, \psi) = 2^{-\frac{E(h(x))}{c(\psi)}} \quad (4.2)$$

Here, $E(h(x))$ is the average path length for instance x across all the iTrees in the forest. The score s is always between 0 and 1, and its interpretation is as follows:

4.1. BACKGROUND AND RELATED WORK

- If the score is close to 1, it indicates a high likelihood of being an anomaly, as the average path length is short.
- If the score is close to 0, it suggests that the instance is likely normal, as its average path length is long.
- If the score is around 0.5, the instance is not clearly distinguishable as either normal or anomalous.

Time Complexity One of the significant advantages of the iForest algorithm is its efficiency, which stems from its low time complexity.

Training Time Complexity: The training phase involves building an ensemble of T iTrees, each constructed from a subsample of size ψ from the original dataset of size n . The time complexity to build a single iTree is approximately $O(\psi \log \psi)$. Therefore, the total training time complexity for the entire forest is:

$$O(T \cdot \psi \log \psi) \quad (4.3)$$

Since both the number of trees, T , and the subsampling size, ψ , are typically small constants that do not depend on the size of the dataset S , the training complexity is effectively linear with respect to the number of trees and subsamples.

Inference Time Complexity: During the inference phase, each instance is passed down each of the T trees to determine its path length. The average path length for an instance in an iTree is on the order of $\log \psi$. Thus, the time complexity for scoring a single test instance is $O(T \log \psi)$. The total per sample inference time complexity is:

$$O(T \log \psi) \quad (4.4)$$

The inference complexity is then linear with respect to the number of instances being evaluated, making the algorithm highly scalable for large datasets.

AUTOENCODER

Autoencoders are a particular type of NN that is trained to reconstruct its input. The formal definition of the problem has been given in [170] which aims to learn two functions, the so-called **encoder** $A : \mathbb{R}^p \rightarrow \mathbb{R}^d$ and **decoder** $B : \mathbb{R}^d \rightarrow \mathbb{R}^p$ that satisfy 4.5:

$$\operatorname{argmin}_{A,B} \mathbb{E}_{x \in \mathbb{X}} [\Delta(x, [B \circ A](x))] \quad (4.5)$$

where Δ is the reconstruction loss function, that usually is the MSE between input and output.

Autoencoders are a relatively mature technology. Like many foundational ideas in machine learning, they were first introduced in the 1990s [98]. Their popularity and effectiveness grew rapidly in the following decades, driven by advances in computational power and the increasing availability of large datasets. In more recent years, several important variations have been proposed, most notably the Variational Autoencoder introduced in 2013 [171].

The most common use of autoencoders is to learn a compact representation of the input. In fact, autoencoders can be seen as a generalization of Principal Component Analysis (PCA) [172]. A second, and more relevant, use for AD exploits Equation 4.5: since outliers are less likely to appear in the training data, they tend to yield higher reconstruction errors. This effect becomes particularly evident when the expressive power of the network is controlled through techniques such as dropout and regularization.

KitNet

KitNet is a Python implementation of Kitsune, originally developed as part of a network intrusion detection system [148].

Its architecture consists of four main components. A feature extractor gathers statistics from the input data stream using a dampening window that gradually forgets older data. A feature mapper then distributes n features to k Autoencoders, ensuring that each receives at most m features through agglomerative hierarchical clustering. An ensemble of k small Autoencoders computes the MSE for each feature group. Finally, an additional Autoencoder processes the k MSE values to produce the overall anomaly score.

KitNet was designed for efficiency and streaming execution. It operates in two modes: **training**, where the AD component updates its weights through a single back-propagation step per input until a predefined number of samples are processed, with an optional update for the feature mapper; and **execution**, where weight updates stop and the model outputs only the anomaly score.

RSHash

Randomized Subspace Hashing (RSHash) [173] is another algorithm that is suited to work with streaming data like KitNet in Section 4.1.1 and so is designed to be fast and with low memory usage.

This model uses an ensemble of size m , and each of the simpler models is composed of w hash tables. Here we report a simplified explanation of how it works since the implementation

4.1. BACKGROUND AND RELATED WORK

details can be found in the original paper.

The first thing that is done is to select a subset of the training, a subset of the features, and some randomized constants that are used in the very next operation for each of the models in the ensemble. Then the points X_i sampled from the training set are min-max normalized into X'_i and transformed into Y_i with:

$$y_{ij} = \begin{cases} -1 & \text{if feature } j \text{ not selected} \\ \lfloor \frac{x'_{ij} + \alpha_j}{f} \rfloor & \text{otherwise} \end{cases} \quad (4.6)$$

Then w hash tables are updated, in more detail at each hash table H_k is associated with a different hash function h_k built in a particular way explained in the original paper [173] and each point Y_i increments the length of the list in cell $H_k(h_k(Y_i))$ by one.

At the end suppose a point X_t to score and $c_k = \text{len}(H_k(h_k(Y_i)))$, take the $\log_2(\min\{c_1, \dots, c_w\})$ if the point were not in the training or $\log_2(\min\{c_1, \dots, c_w\} + 1)$ otherwise for overfitting reasons and to treat all the points the same way (if a point is in the training set the minimum returns at least one) and the score is simply the average value from the ensemble's models. It's easy to understand that an anomalous value would obtain a low count in the hash table and so a low score.

DEEP SVDD

Deep Support Vector Data Description (Deep SVDD) [174] is a recent algorithm introduced as a NN model trained with an AD-inspired loss function. Other models use the outcome of training only indirectly to measure anomalies. For example, Autoencoders are trained to reproduce their input, and since outliers are rare, they usually end up with a large reconstruction error.

At its foundation, Deep SVDD builds on the classical kernel-based one-class classification method, the same idea used by OCSVM. The main innovation is that the kernel is approximated by the NN while minimizing the volume of the hyper-sphere that contains all projected training samples.

Once the optimal matrix of weights W^* is found, the anomaly score for an input x_i is

$$\|\phi(x_i; W^*) - c\|^2,$$

and the loss function used to train Deep SVDD is:

$$\min_{R, \mathbb{W}} R^2 + \frac{1}{vn} \sum_{i=1}^n \max\{0, \|\phi(x_i; \mathbb{W}) - c\|^2 - R^2\} + \lambda^2 \sum_{l=1}^L \|W^l\|_F^2 \quad (4.7)$$

where:

- The first term minimizes the radius R of the hyper-sphere centered at c in the latent space.
- The second term allows a soft boundary, meaning that some points can fall outside the hyper-sphere. The hyperparameter v controls the trade-off between the number of violations and the volume of the hyper-sphere. The set $\mathbb{W} = \{W^l : l = 1, \dots, L\}$ contains the weights of the NN.
- The third term is a regularization penalty on the weights.

4.1.2 INTERPRETABILITY

In many scenarios, it is not sufficient to simply trust the output of a machine learning model. This is especially true when decisions based on the model involve significant risks, costs, revenues, or safety concerns. Consider, for example, a model that determines whether an individual should receive a loan. Since machine learning models can be effected by biases, including racial bias [175], underrepresented groups may be systematically penalized. To mitigate such risks, it becomes necessary to understand how inputs influence the model output (interpretability) and how the internal parameters justify a decision (explainability).

Interpretability and explainability are highly desirable properties in most applications, although they are not universally beneficial. As noted in [157], full transparency can create opportunities for strategic behavior. In the case of a loan approval model, complete visibility into its decision logic could allow applicants to exploit its weaknesses. If the model penalizes individuals who hold more than two credit cards, returning a credit card could artificially increase the likelihood of approval without reflecting any genuine change in creditworthiness.

In practice, interpretability is used to evaluate machine learning models according to several key principles [176, 157]:

- **Fairness:** Ensuring that predictions are unbiased and do not discriminate against underrepresented groups. An interpretable loan model, for example, makes it easier to detect whether decisions are influenced by demographic features.

4.1. BACKGROUND AND RELATED WORK

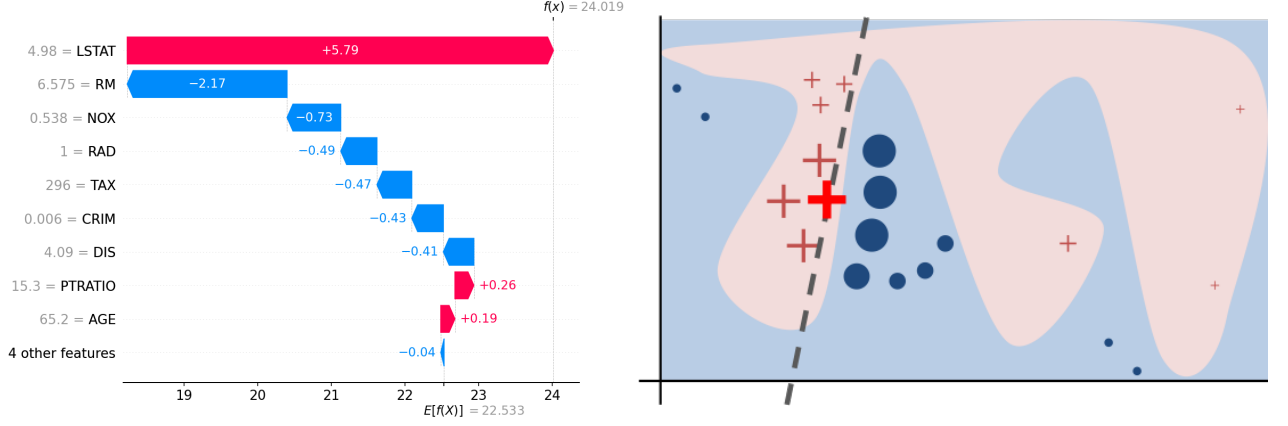
- **Privacy:** Guaranteeing that sensitive information in the data remains protected.
- **Reliability and Robustness:** Verifying that small perturbations in the input do not lead to large fluctuations in the output.
- **Causality:** Checking that predictions rely on causal, not spurious, relationships.
- **Trust:** Users are more likely to trust a system when it provides understandable reasons for its decisions.

In simple models such as linear regression, interpretability is straightforward because inputs and outputs are linked by a linear relationship. With more complex models, such as neural networks, it becomes significantly harder. Some interpretability algorithms are model-specific, exploiting the internal structure of a given method (e.g., for isolation forest [177, 178]). Others are model-agnostic, designed to work with any type of model.

A well-known model-agnostic approach is Local Interpretable Model-Agnostic Explanations (LIME) [179]. LIME perturbs the input instance, trains a local linear model on these perturbations, and weights them by their similarity to the original sample (Figure 4.1b). While effective, LIME is computationally expensive and limited to local explanations.

A more recent alternative is SHAP [180], which builds on similar perturbation principles but employs Shapley values from cooperative game theory [181]. SHAP provides additive explanations that can be both global and local, and it is typically more efficient to compute (Figure 4.1a). SHAP also quantifies the contribution of each feature in both positive and negative directions, offering a clear decomposition of the model output.

The field of interpretability is evolving rapidly due to its importance in ensuring accountability and trustworthiness. In [182] the authors showed that human-machine collaboration can outperform either humans or models acting alone. Achieving such collaboration requires interpretability, highlighting its central role in modern machine learning applications.



(a) SHAP's additive explanations, their sum is $f(x)$ in (b) LIME's local linear model. The bigger cross is the top right corner (adapted from [183]). The positive point to explain (adapted from [184]).

Figure 4.1: Examples of representations of the interpretability algorithms.

SHAP

SHapley Additive exPlanations (SHAP) [180] is a widely used model-agnostic interpretability method grounded in the concept of Shapley values from cooperative game theory [181]. It computes the fair contribution (importance) of each input feature to the model's output through an efficient approximation of Shapley values.

Formally, let $f : \mathbb{R}^F \rightarrow \mathbb{R}$ be a model with F input features, and let $x \in \mathbb{R}^F$ be an instance. The Shapley value for feature i is defined as

$$\phi_i(f, x) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(F - |S| - 1)!}{F!} [f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)], \quad (4.8)$$

where $N = \{1, \dots, F\}$ is the set of all features, S is every subset of features excluding i , and $f_S(x_S)$ denotes the model restricted to the feature subset S . The weighting term ensures fairness by considering all possible feature orderings.

The explanation is then given as a linear additive model

$$f(x) = \phi_0 + \sum_{i=1}^F \phi_i(f, x), \quad (4.9)$$

where $\phi_0 = \mathbb{E}[f(x)]$ is the expected output of the model over the training distribution, and $\phi_i(f, x)$ is the contribution of feature i for instance x .

4.1. BACKGROUND AND RELATED WORK

Computing exact Shapley values is exponential in F , which makes it infeasible for high-dimensional inputs. Kernel SHAP [180] addresses this by approximating the Shapley values through a weighted linear regression (modified version of linear LIME). Specifically, it optimizes

$$\min_{\phi \in \mathbb{R}^F} \sum_{z' \in \mathcal{Z}} \pi_x(z') \left(f(h_x(z')) - \phi_0 - \sum_{i=1}^F \phi_i z'_i \right)^2, \quad (4.10)$$

where $z' \in \{0, 1\}^F$ is a binary vector indicating which features are included, $h_x(z')$ maps z' to a sample in the original input space, and $\pi_x(z')$ is a weighting kernel that guarantees consistency with the Shapley axioms.

Kernel SHAP thus provides an efficient and model-agnostic approximation of Shapley values, making it suitable for complex models such as tree ensembles or neural networks. In the context of anomaly detection, SHAP enables not only the identification of anomalous samples but also the attribution of anomalies to specific sensor signals, thus supporting root cause analysis.

4.1.3 RELATED WORK

As already mentioned, the literature on industrial AD is extensive and diverse, and we have cited some of the most relevant works. Often, methods that prove successful in one domain can be adapted to another with only minor modifications.

In [185], the authors compared PCA and KICA-PCA (an extension of PCA already applied in this context by [186]) with the extended isolation forest [187] to detect anomalies in HPP sensor data. In [188], the authors applied variational modal decomposition and deep convolutional Autoencoders (using multiple convolutional layers) for AD on HPP data. Their features included the OS and the oil level in the tank. While the latter is available, it is not directly related to vibration monitoring but rather to overall maintenance of the unit.

Another important research direction involves the use of interpretability algorithms in AD models to enable RCA. One of the first contributions in this line is [138], where the authors compared permutation-based importance with Depth-based Feature Importance of Isolation Forest (DIFFI) [189], a model-specific algorithm for tree-based models. This work is particularly relevant here, as the local formulation of DIFFI is later employed in Chapter 6 to properly weight the influence of new and old trees.

Concurrently with our work, two publications further validate the feasibility of our approach. In [190], the authors trained a stacked autoencoder for classification and then applied kernel SHAP to explain the results, showing the flexibility of SHAP in supervised tasks. In

[191], the authors tested several AD models—including KNN, Minimum Covariance Determinant, Local Outlier Factor, OCSVM, iForest, and Histogram-Based Outlier Score—and then applied SHAP to perform RCA on rotating machinery. The key difference with our work lies in both the models tested and the methodology. Standard rotating machinery represents a much simpler setting compared to hydroelectric plants. Such systems are often mass-produced, smaller, more controllable, and less strategic in energy production. Data can sometimes be obtained directly from the manufacturer or through controlled laboratory testing, allowing the authors in [191] to apply SHAP to both RCA and fault identification. In contrast, hydroelectric plants are custom-built, have long life cycles, and lack practical ways to intentionally generate faulty data. Additionally, the dynamic influence of water alters system behavior in complex ways that are difficult to simulate accurately.

To this day our work remains one of the few applications of unsupervised AD with interpretability to perform fault detection and root cause analysis in the context of vibration monitoring in HPP.

4.2 DATA

This section presents the dataset used in the case study, addressing quality issues, listing the available features, and describing their relationships with the detectable anomalies. Background information and terminology are provided in Sections 2.3 and 4.1.

4.2.1 DATA QUALITY

Data from HPPs present several quality issues common to industrial environments, as discussed in Section 2.5.2. These issues are further compounded by seasonality effects that depend on the type of plant and its geographical location. These issues shaped the experimental setup.

Data scarcity: Although the system produces a continuous flow of data, the monitoring system was recently commissioned at the time of collection, resulting in a relatively small dataset. This situation is typical in hydroelectric applications, where digitization has only recently been introduced. As a reference, the digitization project described in [192] for Norwegian-Swedish HPPs began in 2018.

Unbalanced data: Many industrial systems usually operate with a long mean time between failures, so anomalies are rare. This is especially true in the hydroelectric, where robust design is necessary due to the strategic role of energy production. To ensure the presence

4.2. DATA

of anomalies, we used data from a unit that the company reported to have experienced maintenance and anomalous conditions.

Unlabeled data: Labeling HPP data is expensive. Subtle anomalies may require weeks of analysis and the involvement of several expert teams. Given the size and dimensionality of the dataset, complete point-wise labeling was not possible. Instead, a list of anomalous events with approximate time spans was available.

High-dimensional multivariate data: The complexity of the system produces high-dimensional data streams, with about 200 features per machine. Vibrations are caused by multiple phenomena, which makes multivariate analysis necessary for fault detection. The features were designed for interpretability rather than independence, leading to redundancy. Except for two features that were constantly zero due to acquisition errors, each feature is linked to specific causes of anomalies, as detailed in Section 4.2.2.

Seasonality: Some effects, such as the vortex rope, depend strongly on the operating point. Hydroelectric companies define OSs, partitions of the operating range where the system behavior is expected to remain stable. These states are not uniformly represented in the dataset because of seasonal variability. When water availability is high, the plant runs steadily at full capacity, so the unit remains mostly in the high-power state. When water availability is low, power is distributed among units to balance inflow, reservoir capacity, and grid demand, which makes the unit operate across multiple states. Transient conditions such as run-up and coast-down are not operating points but transitions between working and inactive state. As shown in Table 4.1, some operating states are underrepresented, which increases the risk that models classify them as anomalies.

Operative states	Number of samples
High power	31174
Low power	30140
Run up	2290
Coast down	1432
Others	4107

Table 4.1: Table with the number of samples for each OS.

4.2.2 FEATURES

We analyzed the vibration data from a single hydroelectric unit. The subset of available features is described below:

- **speed**, **wicked_gate_opening**, and **power**: describe the operating state.

- **1x_mag_bearing, 2x_mag_bearing** (x and y directions): power spectrum of the first and second harmonics, related to shaft unbalance and misalignment issues (see Section 2.3). The 1x features are also called "synchronous".
- **1x_ph_bearing** (x and y directions): phase of the spectrum at the fundamental frequency, mainly used to infer the shaft orbit shape when influenced by imbalance and misalignment.
- **vortex_bearing** (x and y directions): power spectrum corresponding to the frequency where the first harmonic of the vortex rope.
- **subsynchronous_bearing**: integral over the frequencies below 1x (see Figure 4.2), capturing effects beyond the vortex rope.
- **asynchronous_bearing**: integral over the frequencies different from 1x (see Figure 4.2). It is primarily used to distinguish bumps from other problems. A bump, approximated by a delta function, has a Fourier transform equal to one across all frequencies. Thus, an increase in asynchronous_bearing strongly indicates a bump, whereas other problems usually cause localized growth in specific frequency bands.
- **RSI_bearing**: rotor-stator interaction as described in [193], producing pressure fluctuations from the interaction between rotating parts (impeller) and stationary parts (distributors). The frequency depends on the product of the number of impeller blades (13) and nodal diameters (2).
- **crest_bearing, pp_bearing**: crest factor and peak-to-peak values of the sensor waveform.
- **gap_bearing**: average position of the orbit, as illustrated in Figure 2.7b.
- **Orbit#_bearing**: features extracted from the orbit graph in Figure 2.7.

In addition to shaft-relative displacement, we also considered features derived from two additional pairs of sensors: one measuring the machine structure vibration and the other measuring the velocity of the bearing vibration. These sets of features are similar to those used for shaft relative displacement, except for the orbit-related features.

4.2.3 ANOMALIES

Starting from the anomalies recorded by the company, we applied the iterative process described in Section 4.3.1 to identify additional anomalies hidden in the complexity of the

4.2. DATA

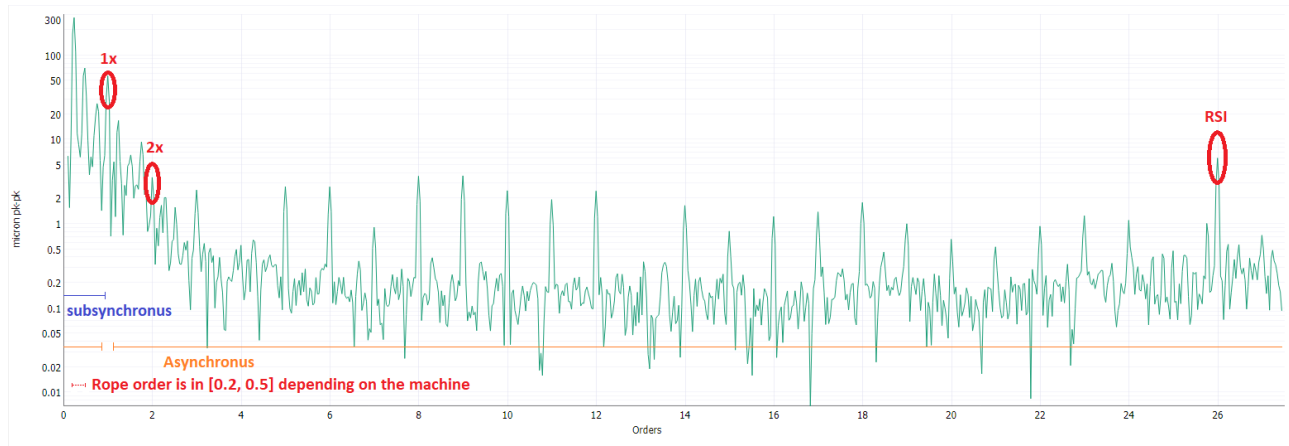


Figure 4.2: Representation of frequencies used to compute the main features (image courtesy of [63]).

data, even though they did not trigger faults. This process produced a set of loosely defined anomalous segments that we employed to evaluate the models. However, it was not possible to establish a precise point-wise ground truth for a rigorous quantitative evaluation.

The anomalies used to evaluate the models are illustrated in Figure 4.3 and include the following:

- **Post-maintenance misalignment:** this anomaly is actually a combination of two effects. The first is related to the shaft components that were more aligned before the maintenance than after, with no real risk but a clear increase in the synchronous spectral components of the vibration. The second effect arises from the bearing being more compliant in one direction, a condition that was corrected during maintenance.
- **January 30, 2022:** a fault occurred that was not directly detectable from the sensors but led to major maintenance.
- **March 22 to April 6, 2022:** an increase in the machine's structural vibrations along the X-axis. As will be better explained in Section 4.5, this anomaly was detected by our models and confirmed by the company after the experiments. This effect was caused by a change in the parameters of the turbine governor which made a structural resonance with Rotor-Stator Interaction (RSI), that is specific to this unit, more pronounced.
- **February 24 to March 13, 2022:** an anomaly known to the company, caused by velometers for structural vibration that were incorrectly installed after maintenance.
- **Recent vibrations and missing data:** In the most recent part of the dataset, the company reported an overall increase in vibrations. At the time of the experiments, the causes

were still under investigation. Additionally, intermittent connection issues occasionally resulted in missing features. Due to the mixed nature of these potential problems, we decided not to analyze this anomaly using SHAP.

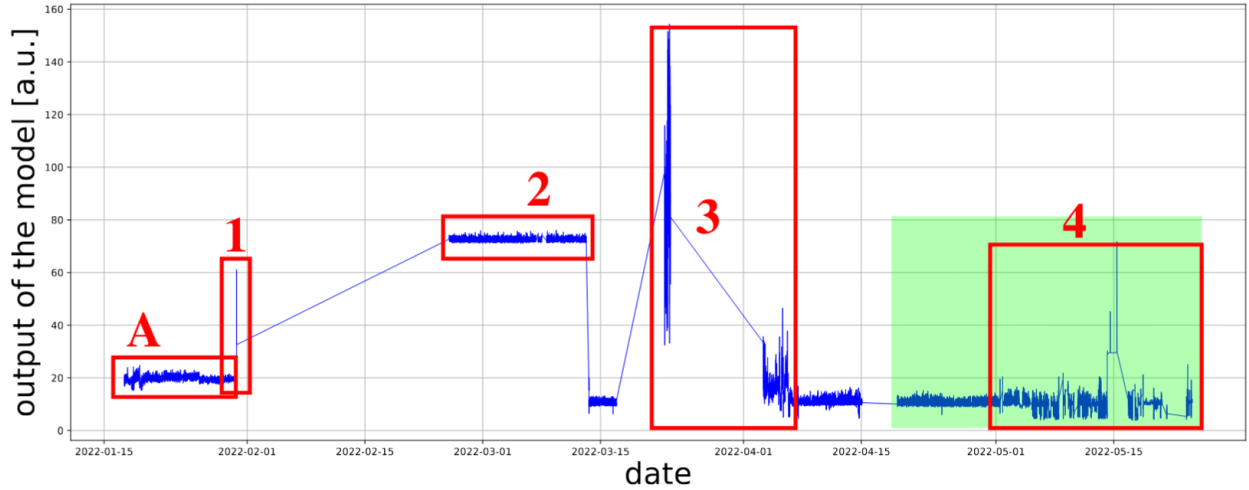


Figure 4.3: Anomalies and Training Data Visualization: The graph illustrates the anomaly scores generated by the Autoencoder model, as an example of what to expect in the results. Regions highlighted in red indicate areas with unusual patterns, either identified from company records or subsequently confirmed by them. The green region represents the training set.

4.3 METHODOLOGY

In this chapter, we describe the methodology and experimental setting used to evaluate the performance of the models presented in Section 4.1.1 on real-world data from a hydropower plant. The design of the experiment was heavily influenced by the data quality issues discussed in Section 4.2.1. The process is similar to the usual ML train-test procedure, except for the evaluation phase, where we relied on domain expert guidance to overcome labeling challenges and identify the most suitable model. After selecting the best model, the relevance of features highlighted by SHAP was assessed by the expert to verify their consistency with the physical behavior of the system.

4.3.1 DESIGN CHOICES

The data quality issues described in Section 4.2.1 make it impossible to apply supervised learning techniques, as labels are uncertain and the data is highly unbalanced. As discussed in Section 2.4, this situation is common in industrial applications, where faults are rare and

4.3. METHODOLOGY

labeling is expensive. Therefore, we approached the problem as an AD task using unsupervised learning methods. The models were trained on a subset of the data representing normal operating conditions and then tested on the entire dataset using the procedure described in Section 4.3.4

4.3.2 PREPROCESSING

The preprocessing procedure was relatively standard, with an additional consideration for rare OS. First, a simple data visualization showed that two features were constant. After confirmation from the domain expert that this was due to an acquisition error in the condition-based vibration monitoring system, we removed them, reducing the total number of features from 65 to 63. Then, since rare OSs can mislead the models by being classified as anomalies, and because run-up and coast-down do not reflect stable operating behavior and have their own analysis procedure, we decided to use only low-power and high-power OSs. This reduced the dataset from 69143 to 61314 samples. Finally, some models such as iForest are not sensitive to feature scale, while others such as Autoencoders are. We did not explicitly scale the values, but the model implementations we used (see Section 4.4) include mean and standard deviation scaling as default preprocessing for the models that require it.

4.3.3 TRAINING

Seasonality (Section 4.2.1) is not in itself a problem, but to build a representative training set it is preferable to use data from the low-water season, when all OSs are present. The examined unit entered the low-water season in mid-April, towards the end of the available data. Hence, a training dataset of 37,668 samples was drawn from that period (refer to Figure 4.3). This should be considered when interpreting the reported anomalies. The misalignment anomaly in the first part of the dataset, before maintenance, is not necessarily a fault if encountered in deployment. In fact, the remaining data show worse shaft alignment, but since the training data come from that later period, the model correctly identifies the beginning as anomalous.

The evaluation procedure described in Section 4.3.4 does not allow for straightforward hyperparameter optimization for each model. This is generally not a limitation in unsupervised AD, where performance depends more on the method itself than on the number of parameters. We only needed to ensure that the number of parameters was sufficient to provide a good representation of the data. The complete list of hyperparameters is reported in Section 4.4.

4.3.4 EVALUATION

Our evaluation focused on two distinct tasks: the AD task and the interpretability task. The first concerned detecting anomalies in the data, while the second dealt with explaining them. Since complete labeling was not available for either task, we collaborated closely with the domain expert.

Anomaly Detection The company provided an initial list of anomalies, but we soon realized it was incomplete, as several anomalous periods discovered during the experiments were not linked to any known fault. To achieve a comprehensive evaluation we adopted the iterative approach shown in Figure 4.4. The process started from the anomalies reported by the company; whenever a model identified new anomalous sections, these were submitted to the domain expert for validation. If confirmed, they were added to the list of anomalies that the models were expected to detect.

Additional evaluation criteria included plausibility, readability, and execution time, as these factors are important for usability in an industrial context. By plausibility, we mean that anomalies should appear as rare events restricted to specific periods of time. In other words we expect models to identify anomalies in well-defined sections of the data rather than scattered across the dataset. By readability, we mean that anomalous and normal sections should be clearly distinguishable from the anomaly score, so that the expert can easily identify anomalous periods through visual inspection. Execution time was loosely considered comparing the order of magnitude of training and inference times to assess whether real-time application is feasible. This is due to the fact that different model families process input data in different ways. NN-based models such as Autoencoders process data in batches, while tree-based models such as iForest follow an instance-wise approach.

Interpretability Interpretability relied entirely on the domain expert. Each fault, either known or discovered during the experiment, was analyzed to identify its root cause. The features highlighted by SHAP as most relevant for AD were then compared to the physical understanding of the fault. If they were consistent from a physical point of view, the model was considered effective at explaining its decisions.

Since SHAP is computationally expensive, it was applied only to the best model identified in the AD task. The algorithm requires a pool of samples to compute Shapley values by comparison. To this end, we used the K-Means utility from the SHAP library with 100 clusters to extract a representative subset of the training data as the background dataset. Depending on the duration of the anomalous period, we also subsampled the anomalous

4.4. EXPERIMENTAL SETUP

section to reduce the number of samples. Finally, we plotted the contribution of each feature to the anomaly score using the SHAP KernelExplainer for each point in the anomalous section (local interpretability). Specifically, we analyzed one point every eighth for the new structural anomaly and for the better alignment before the maintenance.

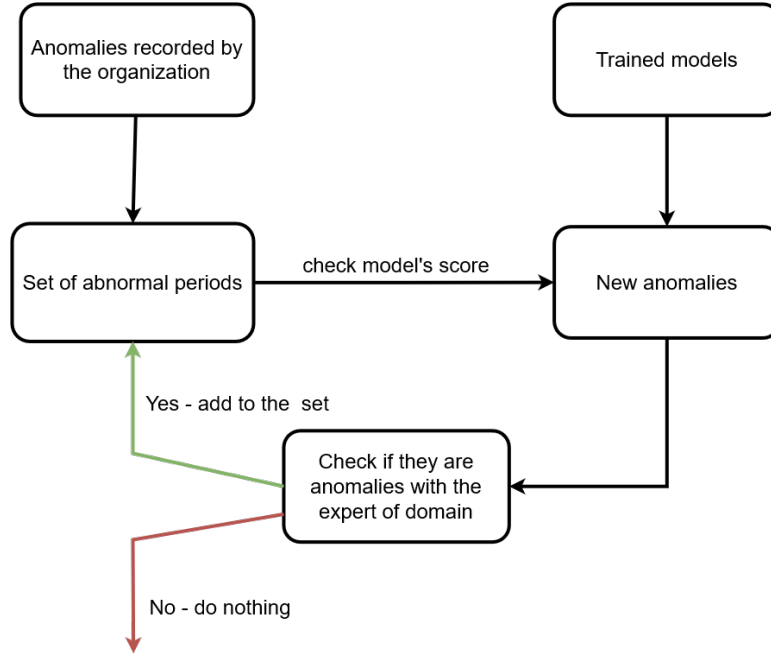


Figure 4.4: Flow chart of the iterative process used to gather the segments of the data that are to be considered as anomalies

4.4 EXPERIMENTAL SETUP

The experiments were implemented in Python [194] using the PyOD [195], PySAD [196], and SHAP [180] libraries. Each experiment was run five times to account for randomness in the training procedure, and average results were reported.

Isolation Forest: Implemented using the PyOD library [195] with 100 estimators and a maximum sample size of 256. The contamination parameter was not relevant, as no threshold was applied.

Autoencoder: Implemented using the PyOD library [195]. The network had five hidden layers with [32, 16, 10, 16, 32] neurons, a learning rate of 0.001, batch size of 32, and was trained for 100 epochs. The Adam optimizer [197] was used with MSE as the loss function, a dropout rate of 0.2, ReLU as the hidden activation function, L2 regularization of 0.1, and linear activation in the output layer.

Deep SVDD: Implemented using the PyOD library [195] with default parameters, except for the output activation function, which was set to linear.

KitNet: Implemented using the PySAD library [196] with default parameters, except for the `grace_period_anomaly_detector`, which was reduced to 4000 to speed up training.

RSHash: Implemented using the PySAD library [196] with 1000 sampling points, decay factor of 0.015, 100 components, and 1 hash function. The method requires estimates of the minimum and maximum values for each feature; we used the values from the training set. Additionally, the model was trained and tested on the entire dataset, as it is a purely stream-based method.

4.5 RESULTS AND DISCUSSION

In this section, we present the results for both the AD models and the interpretability algorithm. We report, for each model, the plot of its anomaly score to analyze which anomalies were detected and which were missed, along with a discussion on the objectives of plausibility, readability and time required for training and inference. Then, we analyze SHAP outputs and compare them with the insights provided by a domain expert.

4.5.1 ISOLATION FOREST

As illustrated in Figure 4.5, the iForest model showed strong performance, detecting all the anomalies. Another strength is its efficiency: as reported in Table 4.2, training requires less than one second and scoring all data points takes under two seconds. The main drawback is that the anomaly scores are somewhat noisy, which makes them harder for personnel to interpret.

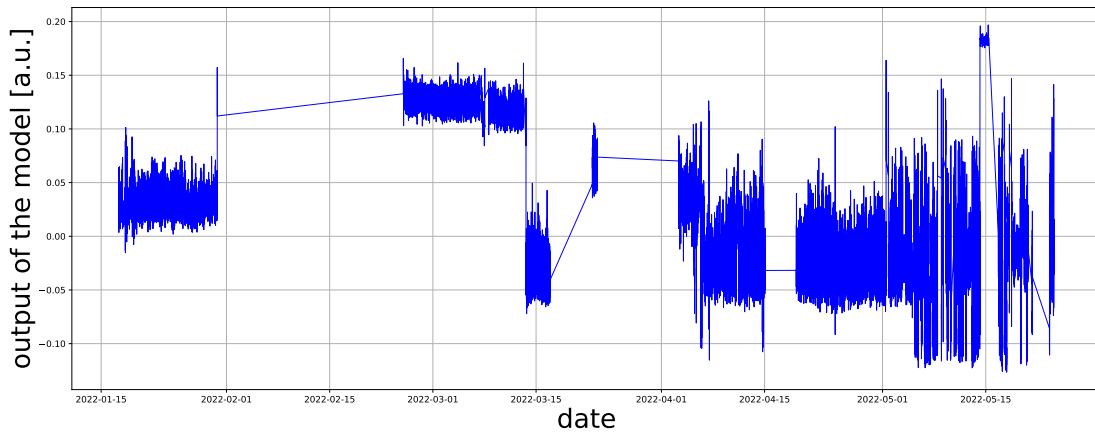


Figure 4.5: Plot of the anomaly score from the iForest model.

4.5. RESULTS AND DISCUSSION

4.5.2 AUTOENCODER

As shown in Figure 4.6, the Autoencoder also detected all the required anomalies, with the additional advantage that its anomaly score has substantially higher values in anomalous periods. This enhanced separation between normal and abnormal data segments makes the model particularly suitable for visualization and monitoring tasks.

Regarding execution speed reported in Table 4.2, training can take up to 300 seconds, which is significantly longer than iForest. Inference, on the other hand, is still very fast, requiring less than five seconds. Overall, this performance is broadly consistent with that of iForest, and even more desirable for visualization purposes.

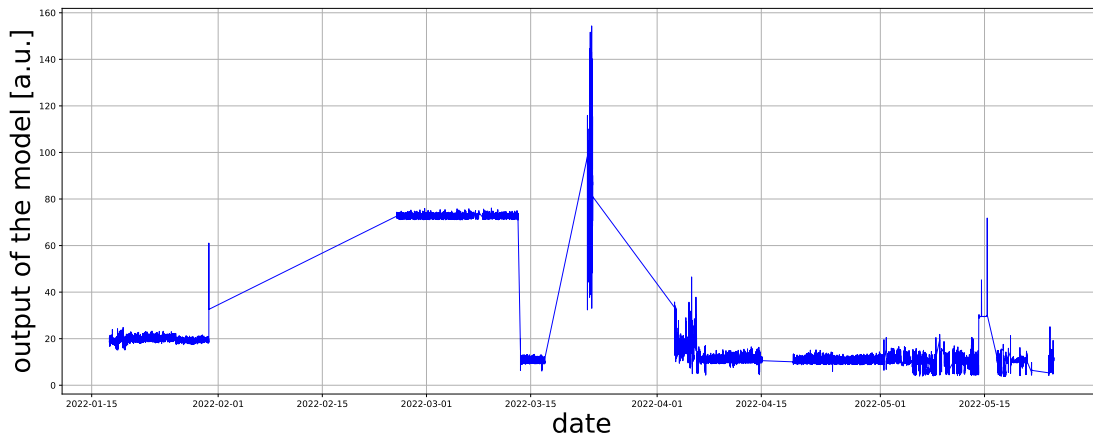


Figure 4.6: Plot of the anomaly score from the Autoencoder model.

4.5.3 KITNET

As illustrated in Figure 4.7, KitNet also detected all anomalies, producing plots that closely resemble those of the Autoencoder. A notable difference is that KitNet tends to amplify certain anomalies towards the end, an effect likely related to its ensemble-based structure.

In terms of execution speed, as reported in Table 4.2, KitNet trains faster than the Autoencoder, with an average training time of about 50 seconds. Inference, however, is more demanding, requiring up to 78 seconds to process the entire dataset.

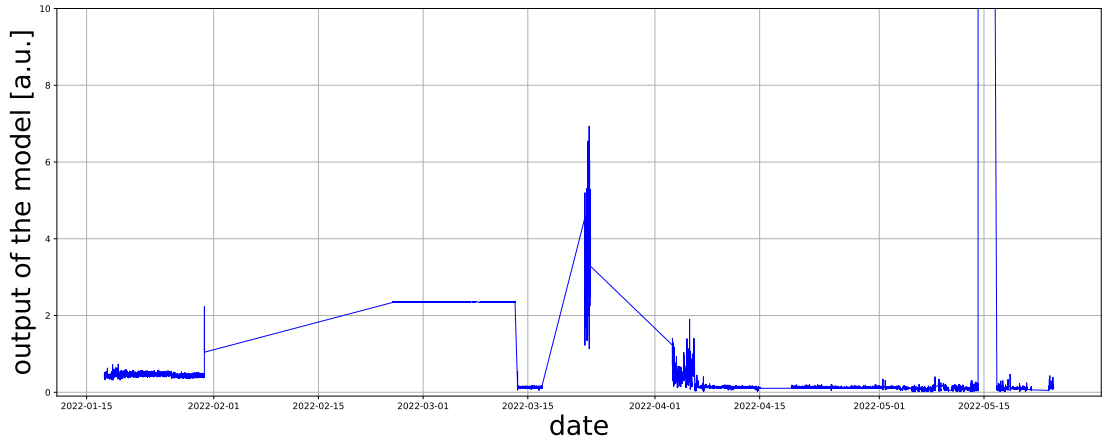


Figure 4.7: Plot of the anomaly score from the KitNet model.

4.5.4 DEEP SVDD

Deep SVDD was not able to detect the lower misalignment before maintenance, as shown in Figure 4.8. This is likely because the model is designed for data streams and does not retain information about persistent anomalies. However, it did successfully identify the other anomalies. Moreover, the anomalies are not grouped together as can be seen in the first half of April where there are spikes instead of a highlighted area.

From the execution time perspective, Deep SVDD trains faster than the Autoencoder but slower than KitNet, requiring up to 200 seconds. During inference, it performs similarly to Autoencoder taking less than five seconds, as reported in Table 4.2.

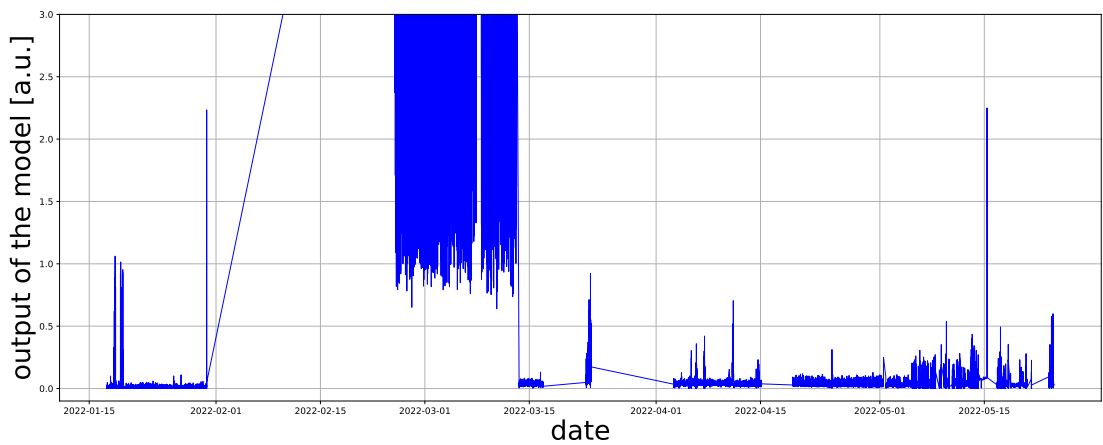


Figure 4.8: Plot of the anomaly score from the Deep SVDD model.

4.5. RESULTS AND DISCUSSION

4.5.5 RSHASH

RSHash is a purely stream-based model, which means it does not retain information about persistent anomalies. As a result, it failed to detect the lower misalignment before maintenance and the structural resonance, as shown in Figure 4.9, and was only able to identify sensor-related anomalies.

In terms of execution time, RSHash is the slowest model, requiring more than 400 seconds to process the entire dataset, as reported in Table 4.2.

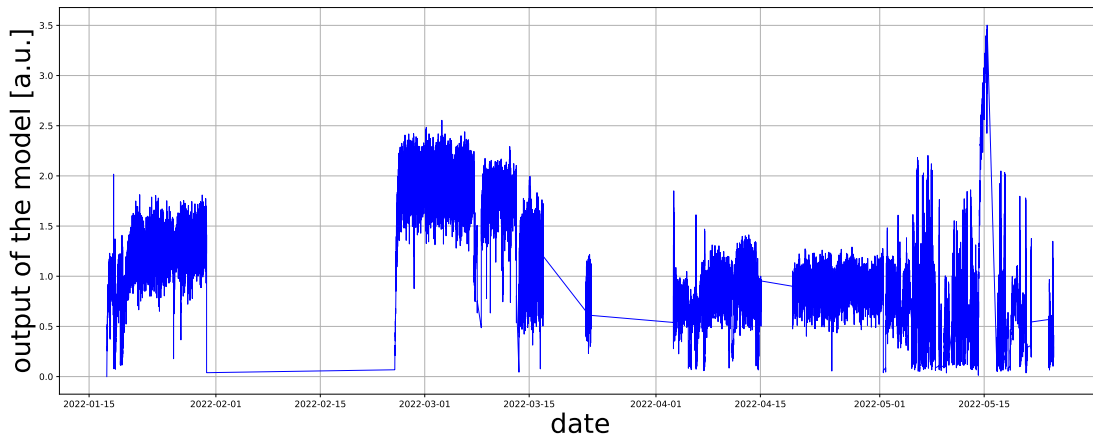


Figure 4.9: Plot of the anomaly score from the RSHash model.

Model	Training time (sec)	Execution time (sec)
Autoencoder	300.685 (25.295 sd)	4.920 (0.402 sd)
iForest	0.329 (0.015 sd)	1.940 (0.038 sd)
KitNet	50.028 (0.479 sd)	77.958 (0.743 sd)
RSHash	446.230 (4.768 sd)	
Deep SVDD	193.560 (18.245 sd)	4.518 (0.901 sd)

Table 4.2: Elapsed times for training and execution with a training set of 37668 points and a total dataset of 61314 points (average of 5 trials).

4.5.6 SHAP

SHAP values are relatively easy to interpret, as they are based on Shapley values, which fairly distribute the model output according to each feature's contribution. Features with values significantly above zero can be considered responsible for the fault, while those significantly below zero are unrelated to the anomaly. Features near zero are less clear, but

when the anomaly score is high, some features inevitably stand out due to the additive property of Shapley values.

Despite its power, SHAP has some limitations. First, interpretability algorithms are designed to highlight a limited number of important features, so when many not-independent features contribute to a fault or when more faults coexist, the weights may be disproportionately assigned to only few features. Second, as a black-box method, SHAP inherits the model's limited understanding of domain-specific properties. The physics of the problem indicates that high values of power features correspond to high risk, but rare low values that are completely safe may also be flagged as anomalous.

POST-MAINTENANCE MISALIGNMENT

Analyzing the improved shaft alignment before maintenance, which the models report as anomalous because the training set is positioned in the latter portion of the dataset, we can see from Figure 4.10 that SHAP highlights the synchronous spectral component of the bearing velocity, but also the gap (average of the shaft orbit) as the main causes. In this case, two unrelated faults overshadow the misalignment. The first comes from the anisotropic rotation of the shaft, caused by the bearing being more compliant in one direction, which produced very large values in `1x_magn_bearing_aX` before the maintenance and returned to normal after. The second is the gap, a fragile feature that mainly reflects the fact that the sensors were not mounted in exactly the same position after maintenance.

Since SHAP is specifically designed to identify a limited number of features that may be responsible for an anomaly, in this case, due to the presence of three superimposed problems, the algorithm can not give a clear indication of the fault's causes. However, as we have seen, it is still useful since the features that it is able to highlight are effectively related to some of the system's anomalies.

ANOMALY OF JANUARY 30, 2022

The anomaly detected on January 30th, as shown in Figure 4.11, initially appeared as a peak involving spectral components related to the bearing velocity and shaft-bearing distance. The collective involvement across the entire spectrum suggests that the anomaly may have originated due to a bump in the system. The peaks in the bearing velocity and shaft-bearing distance crest factors are connected to the subsequent increase in vibrations.

The SHAP algorithm has proven effective in pinpointing the location of the problem at the bearing. Despite the complexity of the issue and the involvement of multiple features,

4.5. RESULTS AND DISCUSSION

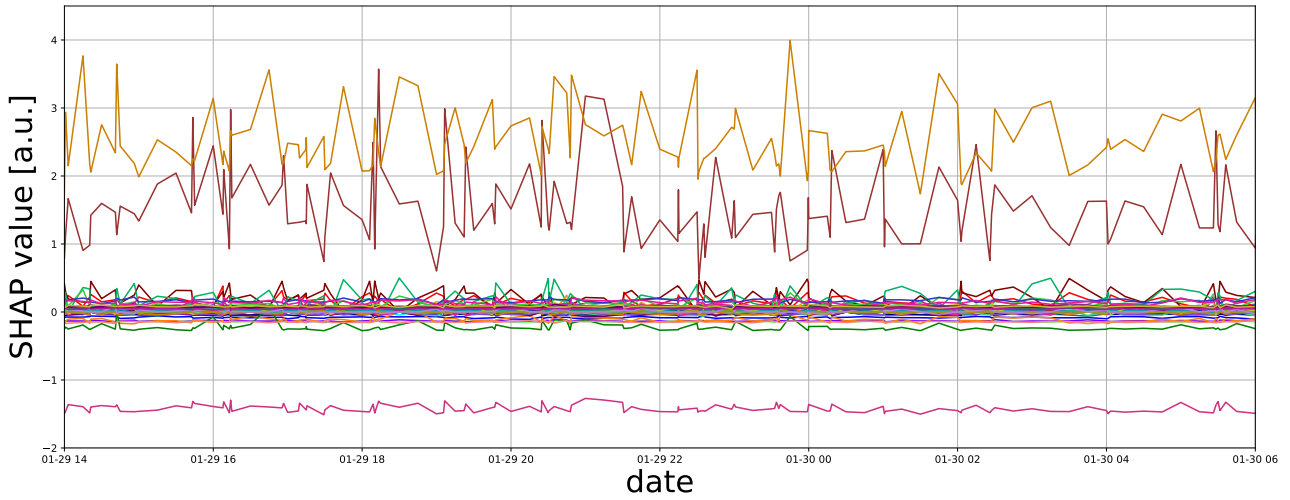


Figure 4.10: SHAP output before January 30th - relevant features: `gap_bearing_rX` (orange line '—') and the `1x_magn_bearing_aX` (maroon line '—').

the results obtained through the use of the SHAP algorithm are completely consistent with the physical explanation of the fault.

UNCONNECTED SENSOR

In this particular scenario, there is relatively little to comment on, as the absence of a signal for shaft-bearing displacement has caused all the related features to fail. As previously stated, the gap feature is not very robust, and in the absence of a signal, it can reach a value that is well above 2500 microns. The reported value is physically impossible because the maximum displacement in either direction is only 500 microns and such a reading would imply the shaft is outside the bearing.

If an expert were to examine Figure 4.12 and subsequently inspect the gap features, they would quickly recognize that the problem is the sensor itself.

NEW ANOMALY

The period between March 22nd and April 6th is the real stress test for this experiment. This anomaly was only detected by our models since no threshold was exceeded, and no report was made by the personnel present at the site. During this period, there was a significant increase in the structure's vibration along the X-axis. Thanks to the collaboration with the domain expert, we were able to obtain a reasonable explanation for this phenomenon.

HPPs are subject to the so-called RSI which refers to the dynamic interaction between the rotating and stationary components of a turbo-machine (the impeller and the distributor). It

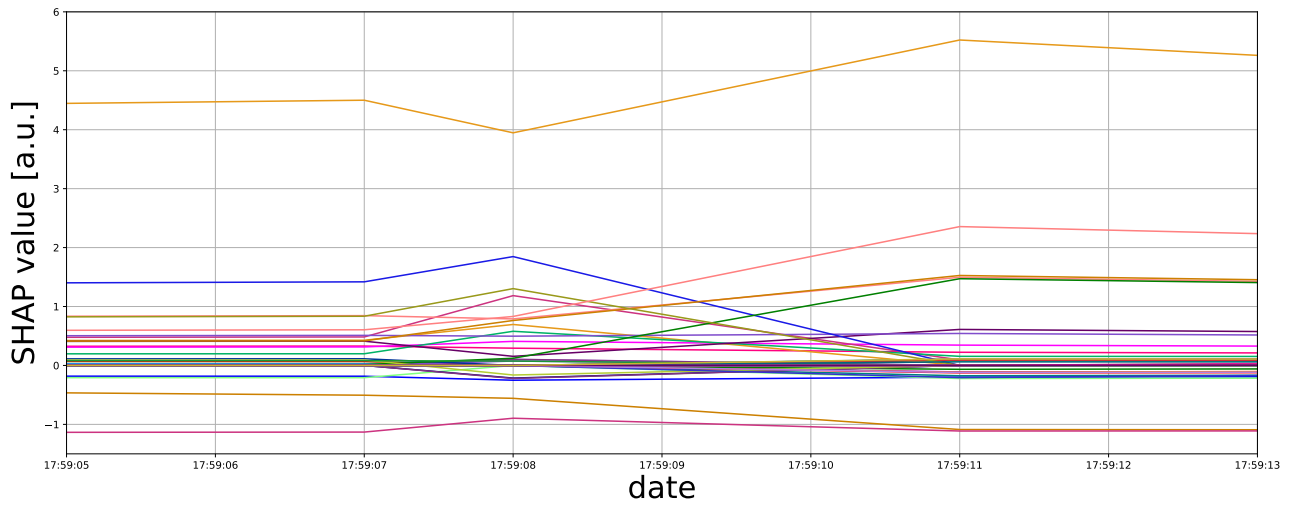


Figure 4.11: SHAP January 30th - bearing's relevant features for the first peak are asynchronous components in X direction (dark blue line '—'), asynchronous components in Y direction (dark yellow line '—') for shaft-bearing displacement. Note that the synchronous spectral component for bearing velocity in the X direction is higher than the ones represented.

can also lead to unsteady flow patterns, pressure fluctuations, and noise. This effect is known to have greater effects when the distributor is in the upper part of its range. Quick changes in the distributor's position can easily affect it and make it increase or decrease in amplitude "around" its steady-state pulsation. The investigated machine has a very particular and unfortunate specificity, in the sense that it runs with a head cover "in resonance" with the RSI pressure field's pulsating forces, thus amplifying the vibration. It happens that the most sensitive direction is the X one. In the period identified as anomalous, a test had been performed at the site for which some parameters were changed in the turbine governor. This led to a much more reactive turbine governor's response to the grid's frequency perturbations, which also made the RSI pressure field more variable. This cause, combined with the onset of the "low-water" season, naturally led to more frequent power set-point variations, and hence to increased turbine governor movements. There is a range of powers (combinations of net head and flow controlled by turbine governor) where RSI effects and vortex rope effects reach a maximum, and this range was much more frequently crossed during the "low-water" season. The continuous changes in the power set-point also cause pressure waves in the penstock, which are reflected in increased low-frequency content (sub-synchronous). Effects of low-frequency pressure field fluctuations are reflected in both bearing velocity and shaft-bearing vibrations, while the RSI effects are noticeable only in the bearing velocity vibrations and might be more easily masked by the normal operation because the head-cover vibrates significantly at RSI frequency in any case.

4.6. SUMMARY OF FINDINGS

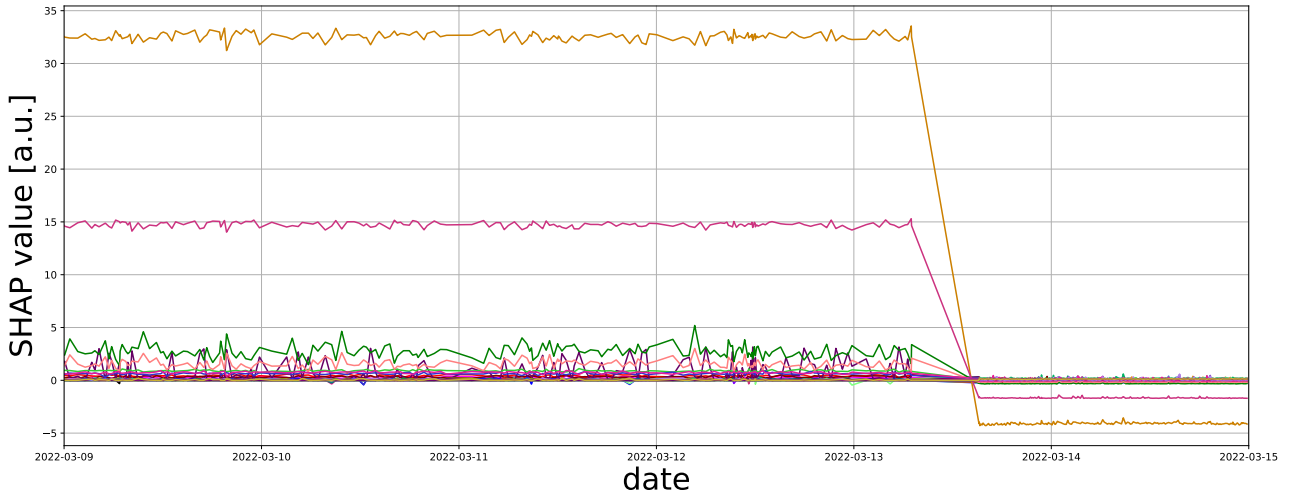


Figure 4.12: SHAP output with unconnected sensor - relevant features: gap on X direction (orange line '—'), gap on Y direction (salmon line '—'), crest on X direction (green line '—'), crest on Y direction (light pink line '—') and synchronous component's phase on X direction (dark magenta line '—'). Where all of them represent the shaft-bearing displacement.

In this particular case, the SHAP algorithm has demonstrated remarkable performance in identifying the root cause of the anomaly, as can be observed from Figure 4.13. It has provided a clear and concise indication of where to concentrate further investigations to discern the underlying causes. It is worth noting that the causes were not readily apparent, and the use of the SHAP algorithm has proven to be extremely helpful in this regard.

4.6 SUMMARY OF FINDINGS

In this project, we have presented a novel method for the PdM of hydroelectric power plants using a combination of fault detection and explainability techniques. Our proposed method aims to speed up the work of domain experts by pinpointing which data needs in-depth analysis and narrowing down the possible causes of the reported anomaly, thus improving the overall efficiency and effectiveness of the maintenance process.

Our method leverages a fault detection algorithm as an additional feature since the data analysis group has to be involved in the decisions, allowing for the identification of anomalies that could potentially indicate an impending failure. Between the ones we tested, iForest and Autoencoder both achieved the desired performance, but we choose the latter since it produces output that is easier to interpret by the user. We then apply the SHAP explainability method to perform root cause analysis.

Our experimental results demonstrate that our method achieves promising performance

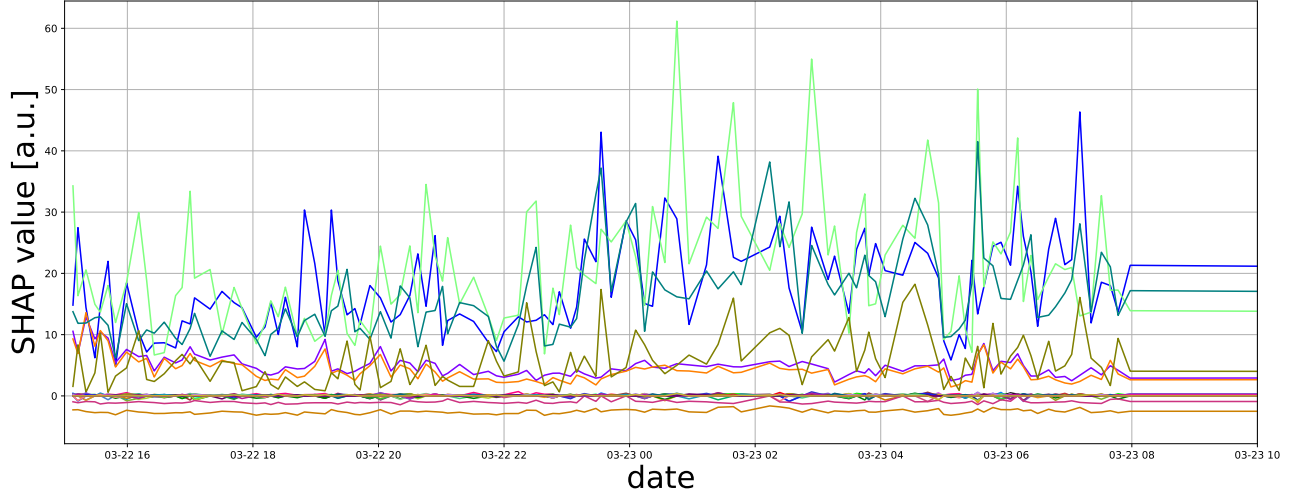


Figure 4.13: SHAP output on the new anomaly - relevant features: sub-synchronous spectral components (teal line '—'), 2x synchronous spectral component (light green line '—'), synchronous spectral component (blue line '—'), vortex rope's spectral component (olive line '—'), asynchronous spectral components (indigo violet line '—') and RMS (orange-yellow line '—') all of them in the structure along the X direction.

in detecting potential faults and identifying their root causes. However, we acknowledge that there is still room for improvement, particularly in the second part where the speed of execution and the quality of the considered features highlighted the limits of the SHAP implementation we used. In our future work, we aim to conduct a study to select an improved set of features, maximizing their relevance for understanding the causes of anomalies while minimizing their correlation. This will allow explainability algorithms to express themselves to their fullest potential. Additionally, we plan to explore different explainability methods, including model-specific ones, to achieve better performance in terms of both speed and accuracy.

Overall, we believe that our proposed method represents a significant step towards more efficient and effective maintenance of hydroelectric power plants, which could lead to improved reliability, reduced downtime, and increased profitability for power plant operators.



Continual Learning for Behavior-Based Driver Identification

Behavior-based driver identification uses driving patterns—such as acceleration, steering angles, and braking patterns over time—extracted from vehicle sensor data to determine the driver’s identity. After traditional authentication methods, including physical and wireless keys, have been proven to be vulnerable [198, 199], researchers have proposed behavior-based approaches in order to tackle the problem [200, 201, 202]. The more recent trend has seen the increasing use of Deep Learning techniques to improve accuracy as well as robustness [203, 204, 205, 206, 207, 208, 209]. These solutions provide significant advantages in solving the ongoing issue of *car theft*, which is still a problem despite many technological advances. Another advantage is that car rental companies and fleet managers can ensure only authorized people drive their vehicles, improving efficiency. It can also enable customized *in-car experiences*, better *roadside assistance*, and optimized *maintenance schedules*.

While the literature extensively explores the potential applications of behavior-based driver identification systems, challenges specifically related to real-world implementation are often overlooked. Many of these systems depend on DL models for classification, which are generally trained offline using large-scale datasets of drivers. Although efficient, collecting enough training data requires a considerable amount of time, leaving vehicles vulnerable to theft during the process [210, 211]. Additionally, in scenarios such as car sharing or rental services, a complete retraining policy is impractical. The model would need to relearn all data whenever a new driver is added, which places unnecessary strain on resources including the GPU and memory. Even with cloud storage, the required time and associated costs can become prohibitive. Lastly, fine-tuning the model using only new data fails to address the

issue, as it can lead to catastrophic forgetting, where the model’s performance on previously learned tasks deteriorates as it adapts to new information [13, 212].

To address these challenges, we propose applying CL techniques to the Driver Identification problem. Since the primary goal of CL is to adapt to incoming data while preserving previously acquired knowledge, it is specifically designed to address catastrophic forgetting while keeping resource usage only marginally above that of fine-tuning [213]. Therefore, we study scenarios where the models learn incrementally from new drivers (see Fig. 5.1 for the general framework). Specifically, we propose an evaluation across three scenarios of increasing complexity and realism, based on the well-known OCSLab dataset [164]. The first scenario represents a classical CL setting, while the other two modify how the data is presented to the system, making them increasingly reflective of real-world conditions. Interestingly, making the simulations more realistic brings both challenges and opportunities. The driver, for example, is not expected to change every few minutes. This continuity is leveraged in this work to introduce two new methods, SmooER and SmooDER, which aim to further narrow the performance gap compared to complete retraining.

This work provides an important step forward in scalable driver identification solutions, demonstrating that CL approaches, such as Dark Experience Replay++ (DER++), can obtain strong performance, with only an 11% reduction in accuracy compared to the static scenario. Furthermore, our novel method, SmooDER, achieves optimal results with only a 2% reduction compared to the 11% of the DER++ approach. Such results prove the feasibility of CL approaches to address the challenges of Driver Identification in dynamic environments, making them suitable for deployment on cloud infrastructure or directly within vehicles. Additionally, we have made our implementation publicly available ¹, for enhanced reproducibility and to facilitate future research in the field.

Our contributions are summarized as follows:

- We investigate the Driver Identification problem in the CL framework.
- We propose and implement on the OCSLab dataset three new scenarios that progressively increase in complexity to incorporate additional challenges and make them more similar to a real application of Driver Identification in a physical vehicle.
- We evaluate the efficacy of CL techniques in the three scenarios, proving the feasibility of their deployment on cloud infrastructure or directly within vehicles.
- We propose two novel methods, SmooER and SmooDER, that exploit the driver ID

¹<https://github.com/MattiaFanan/CL-DriverIdentification>

continuity expected in deployment, achieving superior results compared to the other CL approaches.

The rest of this chapter is organized as follows: Section 5.1 introduces the Driver Identification problem as a proxy task for Fault Detection and Identification in HPPs. Section 5.2 reviews the literature on the Driver Identification problem and the CL framework. Section 5.3 describes our proposed scenarios, the CL approaches we tested, and our new methods, SmooDER and SmooER. Section 5.4 details the experimental setup, including the dataset, training configuration, and model hyperparameters. Section 5.5 presents the results for each scenario. Finally, Section 5.6 discusses the findings and suggests future research directions in this field.

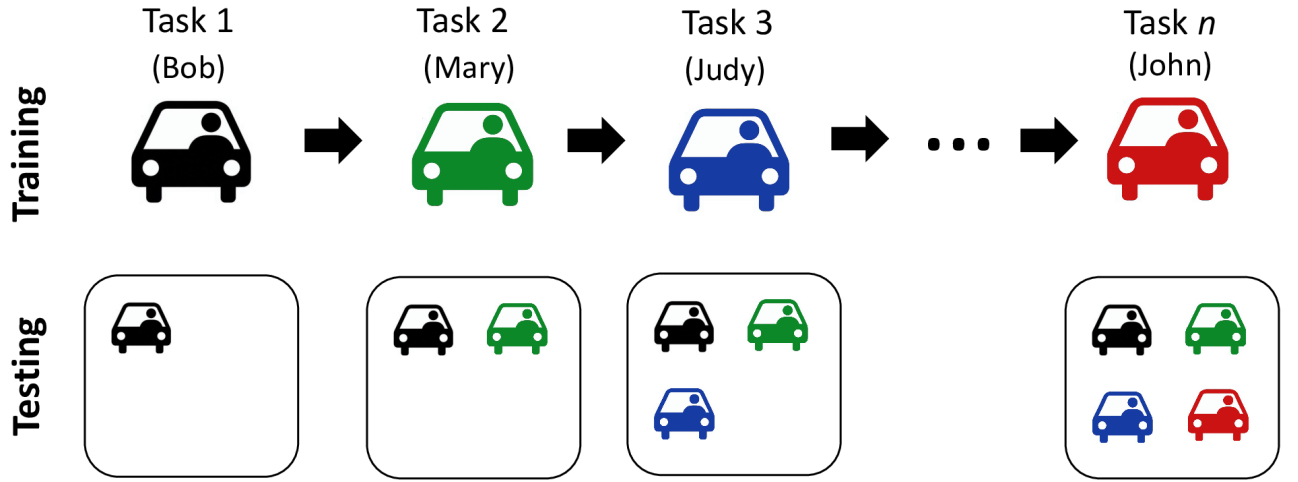


Figure 5.1: General representation of the experimental setup. Since the goal is to learn new tasks as they arrive while retaining knowledge from previous ones, the test set for each task consists of all the test data from drivers whose labels already appeared.

5.1 PROXY TASK

In this work, we use Behavior-based Driver Identification as a proxy task for Fault Detection and Identification in HPPs. The two tasks share strong similarities: in driver identification, the goal is to recognize the driver from their driving behavior, while in fault identification, the goal is to recognize the fault from the behavior of the hydroelectric unit. In both cases, the behavior (the ML model input) is represented by time series data in which the target shows temporal persistence. Drivers do not change every second, and faults typically develop over time and remain until resolved.

5.2. BACKGROUND AND RELATED WORK

Another common aspect is the growing number of possible targets and the presence of resource constraints that discourage a complete retraining of the model. We cannot expect a family to visit the dealership whenever a new driver is added, nor can we expect a plant owner to send all operational data to the manufacturer and wait for a new model each time a new fault is identified. Although some owners choose to use real-time cloud analysis services, this approach is costly and requires strict security measures. An edge computing solution, similar to the in-vehicle computing that we demonstrated to be feasible for driver identification, offers a viable alternative.

Finally, both tasks exhibit, as distinctive feature, a strong interaction between the system and its user. Each driver has a personal driving style that adapts to the specific vehicle, for example adjusting braking behavior to account for differences in responsiveness across cars. Likewise, each fault leaves a distinctive signature in the sensor readings, but hydroelectric units are custom-built and differ in resonance frequencies, grid frequency, and actuator configurations. An example of this is the high structural vibrations discussed in Chapter 4 (see Section 4.5), which were detected in a single unit of one plant. With the CL approach proposed in this chapter, once such a unique fault is identified and analyzed, the model can continue to recognize it automatically from then on.

5.2 BACKGROUND AND RELATED WORK

In Section 5.2.1, we review the literature on behavior-based driver identification. Following that, Section 5.2.2 will provide an overview of the Continual Learning topic, highlighting the main scenarios and methods in the field.

5.2.1 DRIVER IDENTIFICATION

A significant amount of research addresses the challenge of identifying the driver based on behavioral data. This has several practical applications, including preventing vehicle theft and improving personalized driving experiences. Collecting and analyzing drivers' behavioral data enables the optimization of driver-based insurance schemes by dynamically calculating risk factors [214, 215]. Moreover, it would enable the personalization of the driving experience by adjusting vehicle settings based on user preferences, as demonstrated in previous studies [206].

Most methodologies gather the Controller Area Network (CAN) bus data via the On-Board Diagnostics (OBD-II) port, utilizing its variety of sensors and actuators to assess the vehicle's condition. While many researchers do not share the datasets they use, most studies

rely on the publicly available OCSLab dataset [164, 216]. It features 54 sensors from the CAN bus and includes data from ten drivers. Other datasets, although less commonly used, incorporate different data types, such as Global Positioning System (GPS) [217] data and stability measurements [218].

Many studies use AI and ML algorithms to tackle Driver Identification. The framework proposed in [219] addresses the problem by analyzing the driving dynamics signals that can be acquired from commercially available vehicles. The data gathered from the sensors is filtered and pre-processed into sliding windows to allow the extraction of meaningful statistical and spectral features. Then, a driver is associated with every window, and these individual predictions are finally aggregated to obtain a unified result for the entire trip. Erzin et al. [220] explore the amalgamation of various features, including pedal pressure, vehicle speed, engine speed, and steering angle, to discern the driver identity.

Various works have employed classical machine learning algorithms, such as RF [218, 221, 222, 217], SVM [221, 222, 217, 223, 224, 225], KNN [216, 203, 221, 222, 217], and K-means [200]. Marchegiani et al. [223] leverage CAN bus data from an electric vehicle to study pedal operation patterns and GPS traces across different drivers on the same route. Their research aims to develop a user authentication system using ML techniques, such as SVM and Universal Background Model.

Furthermore, there is increasing emphasis in the literature on the use of RNN [206, 207, 226, 204], including Long Short-Term Memory (LSTM) [208, 201, 209, 206]. Their widespread adoption underscores their effectiveness in capturing temporal dependencies and modeling sequential data, thereby improving the accuracy and reliability of behavior-based driver identification systems. Xun et al. [205] propose driver fingerprinting, utilizing CAN bus data for real-time user authentication. Their method employs a combination of CNN and Support Vector Domain Description to detect unauthorized drivers. The authors in [227] introduce an unsupervised method using a three-layer non-negativity-constrained autoencoder to find the best sliding window size. They then employ a deep autoencoder to extract hidden driving behavior features for better driver identification.

5.2.2 CONTINUAL LEARNING

Commonly, a ML model is built on the premise that the training dataset is representative of the reality one intends to model. However, in many cases not all variables at play are observable. Accordingly, if the observable data-generating distribution of the environment changes, the consequence is a performance decay during the deployment. Collecting another dataset and re-training the model is not a definitive solution if the previously acquired

5.2. BACKGROUND AND RELATED WORK

knowledge is not preserved. In fact, additional training on new data (fine-tuning) causes the effect known as catastrophic forgetting, where the model’s performance on previous data deteriorates [13, 212]. Continual Learning overcomes this limitation by enabling models to learn from new data without forgetting previously acquired knowledge.

CONTINUAL LEARNING SCENARIOS

As described in [228] CL is a subfield of ML concerned with incrementally acquiring, updating, and retaining knowledge over a sequence of tasks presented over time. However, the wide variety of what constitutes a valid task sequence often makes it challenging to compare results across different studies. To solve this issue, many works organized similar experimental setups into the so-called CL scenarios. In more detail, a scenario is set by defining the aspect of the problem that is allowed to change between the tasks, the available input information and the expected outputs. Although in the literature one can find many scenarios that derive from the specific needs of the authors that proposed them, we can safely say that the vast majority of the published works use one of the following: *Task Incremental Learning* (TIL) [229] imposes no constraints on the sequence of tasks. Its main limitation is the requirement for a task identifier both at training and deployment time. When such information is not available one can rely on either *Class-Incremental Learning* (CIL) [229] or *Domain-Incremental Learning* (DIL) [229]. The first only allows for disjoint sets of classes. In contrast, the second requires tasks with the same set of classes but allows a domain shift. Lastly, another important scenario to model more complex environments is *New Instances & Classes* (NIC) [230], which allows updating the model knowledge with both the introduction of additional classes and domain shifts (without the availability of the task identifier).

CONTINUAL LEARNING APPROACHES

In the CL literature, the methods can be grouped into three big families known as rehearsal-based, regularization-based, and architecture-based.

Rehearsal-based techniques rely on storing and reusing past data samples during training. While various approaches exist in this category, one of the most renowned methods is Experience Replay (ER) [231], also known as Replay. With this method, samples from the Replay memory (which stores data from previous tasks) are combined with data from the new task. This allows the model to retain accumulated knowledge by periodically revisiting past task instances while learning new information. It has demonstrated remarkable performance in multi-class classification problems under all three CL scenarios (DIL, CIL, TIL).

Regularization-based approaches incorporate additional constraints or penalties during

training to preserve the knowledge of previous tasks. Two well-known methods belonging to this category are Elastic Weight Consolidation (EWC) [232] and Learning without Forgetting (LwF) [213]. The idea behind EWC is to assign higher penalties to changes in parameters that are important for previously learned tasks, thus protecting them from significant updates during the training of the next tasks.

LwF is based on the concept of *Knowledge Distillation*, introduced by Hinton et al. [233]. The loss function incorporates both the classification error for the current task and a distillation loss, which uses the previous model's output as soft labels. This approach encourages the model to produce consistent outputs across both old and new tasks, helping it retain prior knowledge while learning new information.

The general goal of **Architecture-based approaches**, instead, is to modify the original model's architecture by adding new parameters, which help redirect optimizer updates away from the old parameters that contain the accumulated knowledge [234, 235, 236]. These methods generally suffer from two major drawbacks. Firstly, most require the task ID at inference time, which is unavailable in our scenarios. More importantly, they operate by design on the structure of the underlying model, making it larger after every task, which also leads to an increase in latency at inference time.

Furthermore, the CL literature explores methods that combine multiple strategies to improve performance and robustness [237]. For instance, Dark Experience Replay (DER) [238] integrates the Replay memory strategy with the use of logits from previous models, as done by LwF.

5.3 METHODOLOGY

In Section 5.3.1, we define the methodological framework designed to integrate Continual Learning into behavior-based driver identification systems. Then in Section 5.3.2, three progressively realistic scenarios are introduced to evaluate the CL methods in the context of driver identification. In Section 5.3.3, we present the CL methods used in our experiments. Finally, in Section 5.3.4, our proposed approaches, SmooDER and SmooER, are described.

5.3.1 PROBLEM DEFINITION

As mentioned, Driver Identification is an important problem with numerous real-world applications. However, despite existing methods achieving impressive results, they struggle to acquire new knowledge efficiently over time, as retraining a new model from scratch using

5.3. METHODOLOGY

all historical data is resource-intensive and impractical. This limitation raises concerns about the feasibility of implementing them in real-world scenarios due to their scalability issues.

In a generic scenario, a model is trained on a sequence of T tasks. Each task t is associated with a dataset $D_t = (X_t, Y_t)$. Each training instance $x_{it} \in \mathbb{R}^{W \times F}$ in $X_t = [x_{1t}, \dots, x_{nt}]$ is a matrix recording the values of the F features of interest across W consecutive time steps. The corresponding labels are stored in $Y_t = [y_{1t}, \dots, y_{nt}]$ with $y_{it} \in C_t$, where C_t is the set of classes for task t .

The objective of the model at task t , f_{θ_t} , is to learn a mapping from the input space to the set of all classes encountered so far:

$$f_{\theta_t} : \mathbb{R}^{W \times F} \rightarrow \bigcup_{i=1}^t C_i \quad (5.1)$$

$$\theta_t^* = \underset{\theta}{\operatorname{argmin}} \sum_{(x,y) \in \bigcup_{k=1}^t \mathcal{D}_k} \mathcal{L}(f_{\theta}(x), y) \quad (5.2)$$

where $\mathcal{D}_k$ denotes the dataset corresponding to task k , and $\mathcal{L}$ is the loss function. Therefore, after training the model for task t , it is expected to continue performing well on previous tasks as well.

5.3.2 SCENARIOS

We propose and study the following three CL scenarios in the context of Driver Identification, each introducing a progressively higher level of complexity and realism. The goal in all cases is to train the underlying model to recognize new drivers accurately while preserving its ability to distinguish those learned previously.

SCENARIO 1 - TWO NEW DRIVERS

The first scenario is the simplest. The stream is created similarly to those in classic CL benchmarks like Split-MNIST and Split-CIFAR10, where the dataset is split into 5 tasks, each composed of data from only two classes, different from those in the other tasks [229]. This scenario replicates the typical conditions under which CL techniques are usually proposed. Therefore, we split the OCSLab dataset into five tasks, each composed of two drivers' data (see Fig. 5.2). Since for each task, two new classes are added, it fits the CIL setting (described in Section 5.2.2).

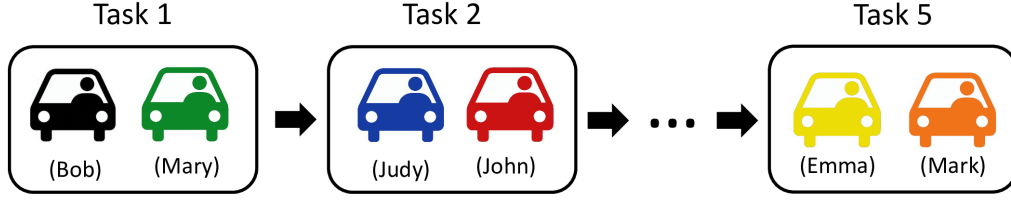


Figure 5.2: Task sequence representation for Scenario 1. Given ten drivers, each task contains information about two new drivers, for a total of five tasks.

SCENARIO 2 - ONE NEW DRIVER

In practice, there is no particular reason to add authorized drivers in pairs like in Scenario 1. Conversely, considering a family car, it is reasonable to expect that the owner might need to authorize an additional driver, such as when a child obtains their driver's license. Therefore, for Scenario 2 we have a sequence of classes as follows: 2,1,1,1,1,1,1, for a total of 9 tasks (see Fig. 5.3). The first task is composed of two classes since evaluating accuracy on a single class at the start would be pointless. Compared to Scenario 1, here more tasks are required to cover the same number of classes (9 instead of 5). As a result, more updates are needed to learn the same number of drivers, which can potentially increase forgetting. Additionally, introducing one class at a time makes it more challenging for the model to learn disjoint representations for each driver, as they are learned separately. Since a new class is added with each task, this scenario fits the CIL setting, as described in Section 5.2.2.

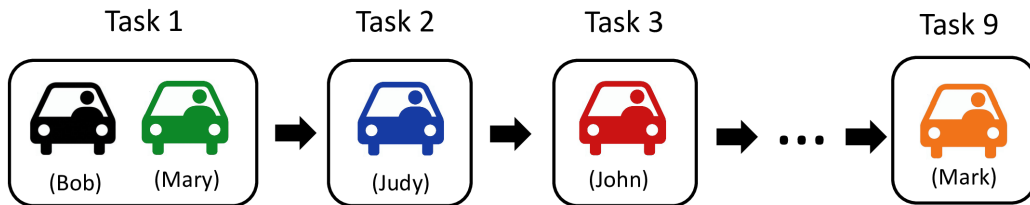


Figure 5.3: Task sequence representation for Scenario 2. A single driver is added to the system each time (except for task 1).

SCENARIO 3 - TWO NEW SESSIONS

Here, we outline a scenario that we believe is the most suitable for the real-world implementation of a driver identification solution.

A critical aspect of adding a new user to the system is the process of collecting the training data, since the behavior-based driver identification system must be temporarily disabled, relying only on conventional identification. Traditional data collection methods, such as

5.3. METHODOLOGY

those used in the OCSLab dataset, typically involve planning lengthy routes for each driver, with multiple sessions needed to accurately capture individual styles. This process is very demanding, often requiring hours of driving, and lacks flexibility. The result is a sequence of driving sessions from various drivers, often with repeated sessions for consistency.

To alleviate this burden, we propose a system in which the conventional and behavior-based authentication components complement each other. When introducing a new user, a startup phase defined by a super-user relies exclusively on conventional identification allowing the collection of enough data to train the model to recognize the new user. After this initial phase, the system can automatically store data from subsequent driving sessions. If the predictions from the conventional and behavior-based identification converge, the collected data can be utilized for additional training to adapt to domain shifts. Conversely, discrepancies can trigger an error notification. The most efficient time to initiate training using the buffered data is when the car is parked, avoiding unnecessary resource usage while driving. This approach can be naturally represented using the OCSLab dataset by segmenting into tasks a random permutation of the driving sessions. Only tasks involving the introduction of a new driver require access via traditional identification methods, enhancing security in the others.

We propose using two driving sessions per task, as depicted in Fig. 5.4. In real-world scenarios, this could represent either a short round trip by a single driver or a longer journey involving a driver change or a stop. Given that the dataset contains 20 driving sessions (two per driver), this setup corresponds to a total of 10 tasks.

Unlike the previous scenarios, which fall under the CIL setting, this scenario aligns with the NIC setting, as described in Section 5.2.2. Here, tasks may include new driving sessions from previously seen drivers or data from entirely new drivers, making it a dynamic and realistic representation of the driver identification problem.

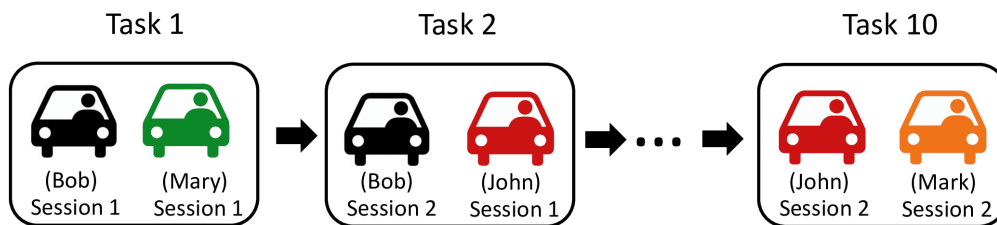


Figure 5.4: Task sequence representation for Scenario 3. The data is divided into drive sessions (there are two of them for each driver) and each task contains the data of two sessions. As a result, the next task can either introduce new drivers or expand the knowledge of previously encountered ones.

5.3.3 METHODS

Our goal is to validate, across three distinct scenarios, whether CL techniques can effectively address the Driver Identification problem. Therefore, we select three baselines: Fine-Tuning is the lower bound as we expect our method to effectively preserve the knowledge, while Joint Training and Cumulative are the upper bounds for model performance. Additionally, we test several well-known CL techniques such as ER, EWC, LwF, and DER.

- **Joint Training:** The model is trained on all classes simultaneously, representing the traditional static training approach. This serves as an upper baseline for CL strategies, as it is not affected by catastrophic forgetting.
- **Cumulative:** This represents an additional upper bound of the CL techniques. The model is trained sequentially on the stream of tasks. However, contrary to Fine-Tuning, the model at each task is trained with all data seen so far. The cumulative approach represents the current solution for adding users; however, it presents scalability challenges.
- **Fine-Tuning:** The model is presented sequentially with data from the current task. No particular technique maintains the model's knowledge, so it is considered a lower bound for the CL approaches.
- **ER:** Experience Replay creates a memory to store samples from previous tasks [231]. We consider a fixed memory size, balancing the space reserved for each class and task. When new data needs to be added to a full memory, a random selection of points is equally removed from each class allocation to make space.
- **EWC:** Elastic Weight Consolidation relies on the Fisher information matrix to identify which neural network parameters are important for the tasks the model has already learned [232]. The Fisher information matrix measures the sensitivity of the loss function to changes in the model parameters. This importance is used to make the important weights more rigid.
- **LwF:** Learning without Forgetting is the most well-known technique among the distillation-based approaches [213]. In our implementation, we compute a distillation loss at each training batch as the MSE between the logits of the current model and those of the frozen model from task $t - 1$. This regularization term is added to the standard task loss.

5.3. METHODOLOGY

- **DER:** Dark Experience Replay [238] is similar to ER as it keeps a small sample from the previous tasks to refresh the old knowledge when learning from new data, but instead of storing the ground truth labels, it stores the so-called "dark knowledge" generated by a previous version of the model (its output logits). The original paper proposes another variant of DER termed DER++, which stores both ground truth and the model output logits for each saved sample.

5.3.4 PREDICTION SMOOTHING

With this work, we introduce two new CL methods derived from DER++ and ER, specifically Smooth Dark Experience Replay (SmooDER) and Smooth Experience Replay (SmooER). These methods are designed to leverage a key feature of the scenarios we consider: the natural continuity of the driver in real-world applications.

We needed a mechanism to correct occasional classification errors due to forgetting without causing significant delays during driver changes. Given that the labels are expected to change infrequently, smoothing the model's output enhances robustness, as multiple consecutive incorrect predictions are needed to significantly alter the integrated result. To achieve this goal we used a causal rolling-window average of the model's output. This approach, inspired by temporal smoothing techniques, ensures that predictions are more stable over short periods of time, thereby aligning better with real-world scenarios where high-frequency label transitions are rare.

Formally, let W be the window size, k the number of classes currently present, and f_θ be the current model. During inference, given the i -th sample x_i , the model calculates its output $z_i = f_\theta(x_i) = [z_{i1}, z_{i2}, \dots, z_{ik}]$, where z_{ij} represents the confidence the model have about $i - th$ sample being a member of the $j - th$ class or not.

We then apply the causal rolling-window average of Equation 5.3 (where W is the smoothing window size) to the outputs, resulting in $\tilde{z}_i = [\tilde{z}_{i1}, \tilde{z}_{i2}, \dots, \tilde{z}_{ik}]$, which ensures that occasional misclassifications do not cause abrupt changes in the output. Finally, the final output is determined by selecting the class with the highest confidence, as shown in Equation 5.4.

$$\tilde{z}_{ij} = \frac{1}{W} \sum_{r=i-W+1}^i z_{rj} \quad (5.3)$$

$$\hat{y}_i = \underset{j \in [1 \dots k]}{\operatorname{argmax}} \tilde{z}_{ij} \quad (5.4)$$

A larger window is advantageous for noisy predictors, as it can suppress more misclassifications. Conversely, a smaller window suits a robust classifier, reducing errors during

driver changes, as fewer new-driver predictions are needed to alter the output.

In this way, our methods, SmooER and SmooDER, exploit the temporal correlation of the driver identifier by incorporating a smoothing technique, making the system more resilient to isolated classification errors while maintaining the independence of the underlying base CL method’s parameters and mechanism.

5.4 EXPERIMENTAL SETUP

In the following we present the experimental setting to evaluate the performance of CL methods in the context of behavior-based driver identification. Specifically, Section 5.4.1 presents the OCSLab dataset along with details of the applied preprocessing. In Section 5.4.2, we describe the machine learning model that forms the foundation of our approach, as well as the details of the training procedure. Section 5.4.3 provides an overview of the hyperparameters for each CL method, explaining their purpose and presenting the optimal values identified in our experiments. Finally, in Section 5.4.4 we describe the metric used to compare the results of our work.

5.4.1 SPECIFICS ON THE OCSLAB DATASET

The dataset used in our tests comprises data from an experiment involving ten drivers (labeled from "A" to "J"), each of them completed two round trips under varied road conditions driving the same car, for a total of about 23 hours of driving. The dataset contains the recording of 51 sensors from the OBD-II port with a sampling frequency of 1Hz (you can refer to the original paper [164] for more detail and the complete list of sensors).

Our main objective is to test the applicability of CL techniques in the Driver Identification problem. Since CL is generally model-agnostic, we decided to build our system over one of the most cited works in this field [226].

To preprocess our data, we removed the features with zero variance in the dataset and then used the mean and standard deviation normalization. This technique standardizes the features of a dataset, transforming each feature x using the formula $x' = \frac{x-\mu}{\sigma}$, where μ is the mean of the feature, and σ is the standard deviation. This process centers the data around zero and scales it to have a unit variance, improving the performance and convergence of many machine learning algorithms.

The next crucial part in preprocessing data for a sequence classification task is to choose suitable parameters for the windowing of our data.

5.4. EXPERIMENTAL SETUP

- The **window length** is how many consecutive samples a training instance is made of and consequently defines the complexity of the task to be learned as well as the richness in the causal relationship that the model can exploit.
- The **stride of the window** refers to the displacement between consecutive windows in the data, allowing for potential overlap. This parameter is crucial as it determines how often each data point appears in the final dataset. A smaller stride means more frequent appearances of the same point in different contexts, enhancing model generalization. However, it also increases the dataset size, which can extend training time. Note that a larger stride does not reduce model responsiveness during deployment, as in the real implementation it is possible to generate a new window with only one new sample.

To set these parameters, we used the configuration from the reference paper with the shortest window length and stride to create a less complex but more responsive model. Specifically, we set the window length to 60 and the stride to 6.

5.4.2 MODELS AND TRAINING

We chose LSTM model because it consistently performed the best in the reference paper's result tables [226]. According to their description, the network structure is a two-layer LSTM, with each layer having 128 hidden states and a dropout rate of 0.5.

LSTMs are sequence-to-sequence models that transform the input into a sequence of hidden states [239]. To create a classifier, a fully connected layer is applied to the last hidden state of the final layer with a Sigmoid activation function. We trained our model with a batch size of 32 using Adam as optimizer [197], following the reference paper, with a standard learning rate of 0.001. The employed loss function is the Categorical Cross-Entropy, commonly used for multi-class classification problems.

The training in our experiments, followed two different setups:

- In the final comparison between the CL algorithms and baselines, we rigidly adhered to the procedure outlined in the reference paper. That is, we used a split of 70% training samples and 30% test samples, training the model for a total of 300 epochs. To avoid overfitting we adopted dropout layers in the model structure as well as early stopping during training.
- To tune the hyperparameters for all CL methods, we sampled half of the test data as validation for early stopping, which was crucial for obtaining results in a reasonable amount of time. A single Nvidia Titan V GPU was used to ensure comparability. With

this setup, many training sessions converged around the 50th or 100th epoch while still providing reliable indicators for the optimal parameters.

Due to varying data points among drivers in the dataset, the number of windows per class was inconsistent (see Table 5.1), which could affect results when changing class order in tasks. To reduce randomness from class ordering and sampling, we ran each experiment with 4 different seeds and 4 class order permutations.

Table 5.1: Count of windows in training and test set for each driver’s drive session.

Driver ID	Session	Train Windows (70%)	Test Windows (30%)
0	1	367	156
0	2	466	199
1	1	773	330
1	2	717	306
2	1	455	194
2	2	409	174
3	1	701	300
3	2	831	356
4	1	509	217
4	2	463	198
5	1	677	298
5	2	596	255
6	1	404	173
6	2	458	195
7	1	567	242
7	2	574	245
8	1	427	182
8	2	472	201
9	1	417	178
9	2	610	261

5.4.3 CONTINUAL LEARNING APPROACHES

Here, we describe the specific hyperparameters considered for each CL approach, their purpose, and the values used in the final comparison. Unless otherwise stated, all methods use the learning rate, optimizer, and batch size settings outlined in the previous section.

In A.1, we provide a detailed explanation of the hyperparameters tuning process, including the reasoning behind the selection of each optimal value.

- **ER:** This method needs two hyperparameters. The first is *Memory Size*, which is also the most important as it is directly related to the resource constraints.

5.4. EXPERIMENTAL SETUP

The optimal value of **1000** was selected for this parameter, as it limits memory usage without causing a significant performance loss compared to the Joint Training baseline. The second hyperparameter is *Replay Ratio*, which governs the mixing ratios of instances from the training set and the memory in a batch of fixed size. Its impact is marginally on the performance and mainly on the time required for training, as the training set is diluted into more and more iterations when the ratio of memory instances increases. Here we chose to mix the two in equal parts, so its final value is **0.5**.

- **EWC and LwF** : They both have a single hyperparameter λ that is used as a weight to balance the influence of their regularization loss before adding it to the training loss. The best values found in our experiments were **10000** for EWC and **5** for LwF.
- **DER++**: This method includes both rehearsal and regularization components. Similar to ER, the more important hyperparameter is the rehearsal

Memory Size. Despite needing slightly more resources than ER to store additional logits, we chose to keep the optimal value at **1000**. The optimal values for the parameters α and β , which balance the losses between memory logits and ground truth labels, were both **1**.

- **SmooDER and SmooER**: The hyperparameters for these methods remain the same as in their original versions, as the smoothing process is independent of the underlying approach. An additional parameter, the *smoothing window size*, is set to six, based on the dataset and data processing characteristics. This choice is aimed at the maximum allowed size to effectively reduce errors while limiting the worst-case delay during a driver change. Given the sampling frequency of 1 Hz and a data windowing stride of one sample during deployment, this choice results in a maximum delay of six seconds.

5.4.4 EVALUATION METRICS

In the following part are defined the metrics used in Section 5.5 to evaluate the performance of all the CL approaches across the three scenarios.

Let's define $D_{train,i}$ as the training dataset for task i . Then $U^{(t)}$ is the dataset composed by the union of all test datasets seen until task t or more formally $U^{(t)} = \bigcup_{i=1}^t D_{test,i}$. Then we define $ACC^{(t)}$ as the accuracy calculated on the test datasets seen so far:

$$ACC^{(t)} = \text{accuracy} \left(U^{(t)} \right) = \text{accuracy} \left(\bigcup_{i=1}^t D_{test,i} \right) \quad (5.5)$$

where *accuracy* in the equation calculates the accuracy of the model given the data.

Then ACC reports the total accuracy calculated after the training on the last task or formally

$$ACC^{(t)} = ACC^{(T)} = accuracy \left(\bigcup_{i=1}^T D_{test,i} \right) \quad (5.6)$$

where T is the last task of the data stream.

Moreover, we define the *gap* as the difference between the model trained on all data together (Joint Training approach) compared to using a Continual Learning approach such as replay, LwF, etc. Formally, let's define as A_{joint} the accuracy of Joint Training as the accuracy on all the test datasets, which represents ideally the upper bound of the CL methods. Then we define the gap as:

$$gap = ACC - A_{joint} \quad (5.7)$$

A lower gap means that the CL method is performing closer to Joint Training, while a larger gap indicates a larger performance loss due to the continual learning setup. The two metrics, ACC and gap, are useful to determine how much the CL approach is close to the upper bound.

5.5 RESULTS AND DISCUSSION

In the following we present the results for each scenario, highlighting the performance of all tested CL methods and comparing these approaches with our SmooER and SmooDER methods.

In Section 5.5.1, we discuss results for the *Two New Drivers* scenario, followed by the *One New Driver* scenario in Section 5.5.2, and the *Two New Sessions* scenario in Section 5.5.3. We provide two figures for each scenario, with the left one comparing the methods and baselines mentioned and the right one focusing on only the best non-baseline methods. A summary table presents the performance of each method in each scenario after learning the last task. Next, we present the results on the effect of prediction smoothing in Section 5.5.4. Finally, we conclude with an overall analysis of the conducted experiments and results in Section 5.5.5.

5.5.1 SCENARIO 1 - TWO NEW DRIVERS

This scenario has the least complexity among the three considered due to its shorter sequence length. As sequence length increases, the effects of catastrophic forgetting are

5.5. RESULTS AND DISCUSSION

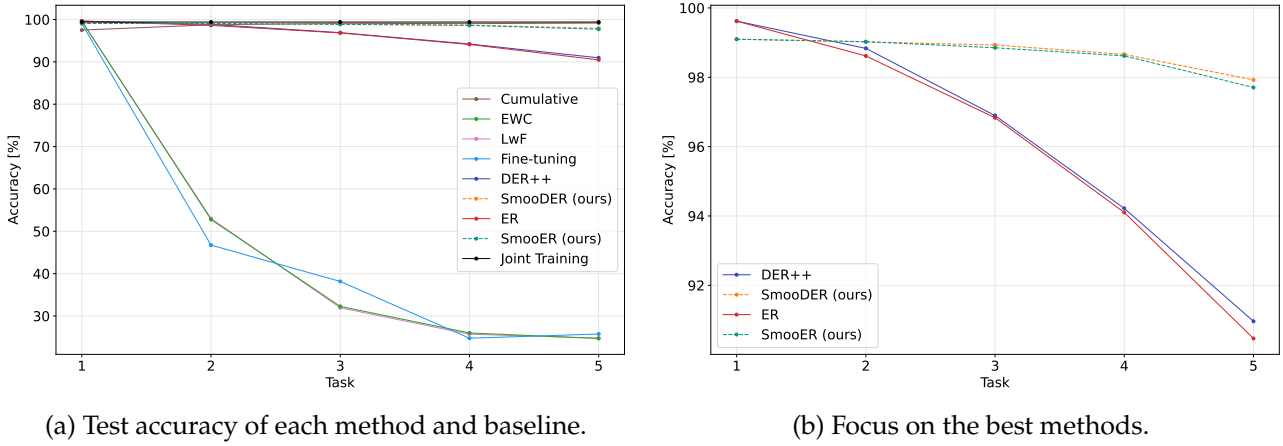


Figure 5.5: Results for Scenario 1.

expected to become more pronounced. Therefore, as reported in Table 5.2 this scenario assessed the highest performance among the three.

From the evolution of the accuracy along the sequence of tasks reported in Fig. 5.5a, we can distinguish three groups of methods: purely regularization-based methods like LwF and EWC, which have no effect in contrasting forgetting as their evolution is very similar to Fine-Tuning. Then, we have ER and DER++, which, instead, demonstrate significantly better accuracy. As expected, DER++ outperformed ER since it is a more advanced method. Specifically, ER achieved 90.47% accuracy, while DER++ 90.96%. Compared the upper bound of 99.4% set by Joint Training (Table 5.2), this translates in a gap of less than 9% for DER++.

Lastly, we have our proposed methods, SmooDER and SmooER, which further enhance resistance to forgetting, achieving approximately 98% accuracy, with a 7% accuracy gain and narrowing the gap to less than 2%. Fig. 5.5b highlights the comparison between our solutions, ER, and DER++, demonstrating their clear advantages, even with a small number of tasks.

5.5.2 SCENARIO 2 - ONE NEW DRIVER

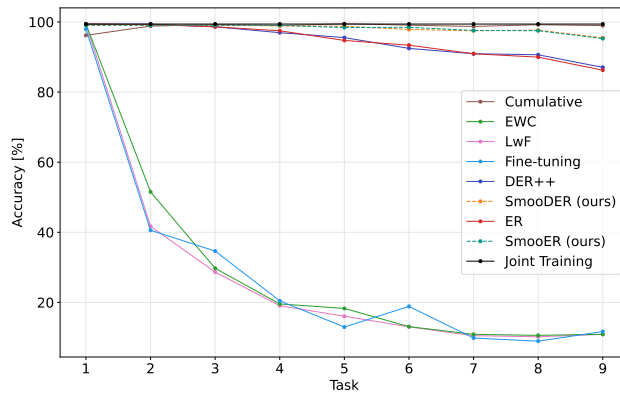
As described in Section 5.3.2, considering a single new driver for each task to increase realism resulted in a more challenging scenario. In fact, we are now dealing with twice the number of tasks. Table 5.2 shows that, as expected, the performance of all methods declined compared to Scenario 1, with the best-performing ones, ER and DER++, experiencing a reduction of around 4% in accuracy. Notably, SmooDER and SmooER demonstrated even lower degradation, with a decrease of only about 2.5%. Given the longer sequence and the complexity of learning each driver individually, this minor drop indicates that our methods

Table 5.2: Performance for each scenario and for each CL technique. The best accuracy for each scenario is highlighted in bold.

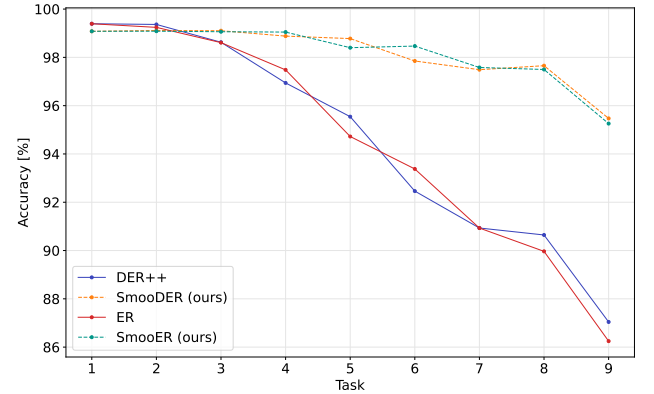
	Scenario 1 Two New Drivers		Scenario 2 One New Driver		Scenario 3 Two New Sessions		Average of all Scenarios	
	ACC $\uparrow$	gap $\downarrow$	ACC $\uparrow$	gap $\downarrow$	ACC $\uparrow$	gap $\downarrow$	ACC $\uparrow$	gap $\downarrow$
Cumulative	99.16	0.24	98.96	0.44	97.43	1.97	98.52	0.88
Joint Training	99.40	-	99.40	-	99.40	-	99.40	-
Fine-tuning	24.55	74.85	10.87	88.53	19.17	80.23	18.20	81.20
ER	90.47	8.93	86.25	13.15	87.08	12.32	87.93	11.47
EWC	24.69	74.71	10.92	88.48	19.10	80.30	18.24	81.16
LwF	24.78	74.62	10.88	88.52	20.01	79.39	18.56	80.84
DER++	90.96	8.44	87.04	12.36	88.87	10.53	88.96	10.44
SmooDER (ours)	97.93	1.47	95.47	3.93	98.22	1.18	97.21	2.19
SmooER (ours)	97.71	1.69	95.26	4.14	97.45	1.95	96.81	2.59

have an acceptably low level of performance degradation as the number of tasks increases.

Fig. 5.6a presents an accuracy progression very similar to Scenario 1, with a notable observation: without effective countermeasures against forgetting, accuracy can drop as low as 11%, as seen with LwF, EWC, and Fine-Tuning. Also, in this more complex scenario, the accuracy gap between Joint Training and DER++ (similarly ER) increases to about 13%. While this gap might still be considered acceptable, it highlights the need for further improvement to mitigate forgetting and enhance performance. In this regard, our methods, SmooDER and SmooER, achieve a final accuracy of approximately 95%, reducing the gap to Joint Training (99.4%) to less than 5%. This translates to an improvement of roughly 8% over ER and DER++ which can be better appreciated in Fig. 5.6b, demonstrating the robustness of our approach in managing forgetting in more challenging scenarios.



(a) Test accuracy of each method and baseline.



(b) Focus on the best methods.

Figure 5.6: Results for Scenario 2.

5.5.3 SCENARIO 3 - TWO NEW SESSIONS

In Scenario 3, complexity is further increased by including partial data from two classes per task, resulting in a sequence of length ten. Here, each task may involve learning a new driver or expanding knowledge of an already encountered one. Despite this added complexity, the performance of various CL methods shows a slight improvement over Scenario 2. This observation provides two key insights: first, it suggests that the primary challenge still lies in the length of the task sequence. Second, the slight performance boost hints that learning multiple drivers per task enhances the model’s capabilities to learn how to effectively distinguish them.

Fig. 5.7a presents a different accuracy progression compared to the other scenarios. This discrepancy occurs because the evaluation is performed on test data from both sessions, even when only the first session has been used for training (see Fig. 5.1). As a result, accuracy tends to improve in the later tasks as the model learns the final session for each driver. This highlights the effectiveness of methods able to counteract forgetting in this context, as they display a *U-shaped* accuracy trend, unlike methods such as LwF and EWC.

Fig. 5.7b focuses specifically on the best methods. ER and DER++ achieve accuracy levels of 87% and 89%, respectively, maintaining a gap of around 10% from Joint Training. Notably, our methods, SmooER and SmooDER, show an even better accuracy, reaching 97.45% and 98.22%, respectively, thus narrowing the gap to the upper bound to approximately 2%. As a final consideration, the random inclusion of domain-shift elements—when the second driving session is introduced for each driver—likely contributes to the approximately 10% performance improvement of methods like EWC and LwF compared to Scenario 2.

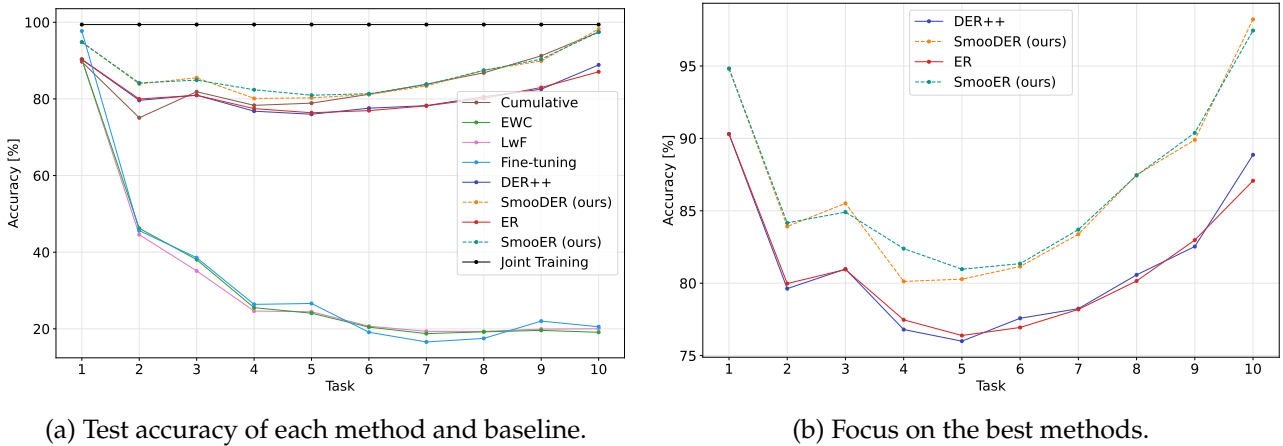


Figure 5.7: Results for Scenario 3.

5.5.4 PREDICTION SMOOTHING

The optimal target model for prediction smoothing must achieve a delicate balance: it should neither exhibit excessive precision, as this could replace misclassification errors with delay errors, nor demonstrate excessive instability, as approaching uniform class probabilities would render any algorithm incapable of extracting meaningful insights.

With the selected hyperparameter configuration reported in 5.3.4, models with base accuracies around 80-90% benefit the most from this smoothing technique. This is evident in Table 5.3, where the ER and DER++ methods show substantial improvements with accuracy gains of up to nearly 9%. On the other hand, models that are either highly accurate or insufficiently accurate experience mixed or negligible changes.

To illustrate the tangible benefits of our approach, see Figure 5.8. In this figure, we simulate the time continuity of a real-world driving scenario by grouping the test set instances by drive session. The double-stair shape represents the order of driver IDs, from 0 to 9, for each of their two recorded driving sessions. On the left, the raw model predictions display numerous misclassifications (spikes). On the right, after smoothing is applied, most of these misclassifications are corrected.

Table 5.3: Performance before and after the prediction smoothing for each scenario and for each CL technique.

	Scenario 1		Scenario 2		Scenario 3		Average Increment
	before	after	before	after	before	after	
Cumulative	99.16	98.83	98.96	98.81	97.43	98.74	0.83
Joint Training	99.40	98.97	99.40	98.97	99.40	98.97	-0.43
Fine-tuning	24.55	24.44	10.87	10.87	19.17	19.36	0.03
ER	90.47	97.71	86.25	95.26	87.08	97.45	8.87
EWC	24.69	24.43	10.92	10.87	19.10	19.12	-0.10
LwF	24.78	27.02	10.88	10.87	20.01	22.57	1.60
DER++	90.96	97.93	87.04	95.47	88.87	98.22	8.25

5.5.5 OVERALL ANALYSIS

In summary, SmooDER consistently outperforms all other methods across all three scenarios. The average results presented in Table 5.2 show that DER++ and ER achieve average accuracies of 88.96% and 87.93%, respectively, thus being more effective than LwF and EWC. The latter methods yield significantly lower accuracies of 18.24% and 18.56%, only marginally surpassing the Fine-Tuning baseline at 18.20%. This can be attributed to their design being less suited to our context, which involves frequent introductions of new classes. Specifically,

5.5. RESULTS AND DISCUSSION

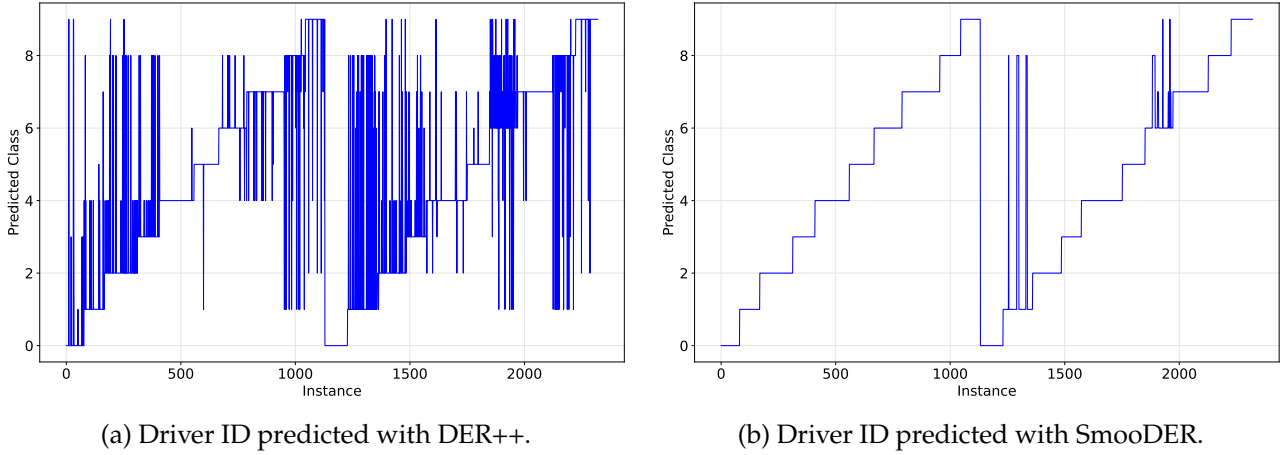


Figure 5.8: Comparison between DER++ and SmooDER. The test set windows are organized by their drive sessions index and driver ID, with SmooDER successfully suppressing most classification errors.

EWC is more effective at addressing domain shifts than at handling challenges related to class expansion, while LwF relies on using current task samples together with guidance from a model trained on earlier tasks. This approach, however, does not efficiently recover information in tasks where, for example, all instances represent newly added drivers, as the teacher model becomes ineffective in such cases.

In contrast, methods that incorporate rehearsal, such as DER++ and ER, use a small yet representative subset of previously seen classes during each task, enabling the model to learn all seen classes jointly. This approach not only prevents forgetting but also enhances learning efficiency. Although these approaches perform well, a measurable gap (10.44%) remains compared to the Joint Training reference, where all data is trained simultaneously. SmooDER and SmooER address this challenge more effectively, reducing the gap to just 2.19% and demonstrating their capability to achieve near-optimal accuracy.

Scalability in terms of both memory and time requirements is another key result, which we have not addressed in previous sections. Fig. 5.9 illustrates the training time (in seconds) for each task across all CL methods in Scenario 3. Among these, the Cumulative baseline, currently used for adding new users, achieves optimal accuracy by retraining from scratch with both old and new data. However, this approach leads to linearly increasing time requirements per each new task. In contrast, Fine-Tuning only processes new data, resulting in nearly constant training time across tasks. LwF, positioned between these approaches, also maintains constant time requirements, as it only computes a regularization term without significant extra computation. Meanwhile, EWC shows a growing time curve, as its implementation requires a separate weight matrix for each task. Alternative versions of EWC, which condense these matrices, are available to reduce training time. Memory-based

methods like DER++ and ER offer a constant time curve, situated between Fine-Tuning and Cumulative, depending on memory size. It is important to note that CL methods have no effect during the first task, so these considerations apply from task 2 onward. Lastly, SmooDER and SmooER require training and testing times similar to DER++ and ER respectively, as the additional computational overhead is minimal.

Finally, our approach has intrinsic modularity, where the underlying model, the CL method, and the prediction smoothing technique do not impose constraints on each other. Therefore, any future advancement on any component can be easily integrated into the system. This makes it possible to extend our work to other fields, as long as the following properties hold. The input should be a time series, and the labels should be generally consistent in time. In human activity recognition, for example, we can employ this approach to enable the training of the model on user-defined activities. When dealing with faults, which are inherently persistent over time, our approach allows for the definition of faults specific to the structure of a particular machine, a characteristic often seen with hydroelectric turbines.

The results of our work show the superiority of our approach. This is due to the purposely tough challenges we put in our scenarios, with many tasks and the rapid introduction of new classes.

Regarding memory requirements, Cumulative approach needs as a training dataset the concatenation of all data from the previous and the current task, so the memory requirement grows linearly in the number of tasks, amounting to a total of 120 MB in our experiments. In contrast, CL approaches use data only from the current task, thereby maintaining a constant memory requirement. Moreover, the highest-performing methods DER++ and ER (as well as their respective variants, SmooDER and SmooER, which are identical in memory requirements) only need to store a small additional subset of samples for rehearsal.

In the results shown in Table 5.2, we use 1,000 samples, occupying just 11 MB — specifically, less than $1000 \times (60 \times 46 \times 4B + 8B + 10 \times 4B) = 11,088,000B$ for DER++ and $1000 \times (60 \times 46 \times 4B + 8B) = 11,048,000B$ for ER. Including the network size of 0.89 MB, a total of only 12 MB of memory is needed. Finally, for the deployment environment, just 6 MB of RAM is sufficient: 0.89 MB for the network, 3.93 MB for the backward pass during training, and 0.35 MB for the input batch. This efficient memory usage enables direct deployment on small devices, such as those commonly installed in vehicles [240].

When considering the eventual implementation of this system, it is important to note that during training, the CL dataset size remains constant, whereas the Cumulative dataset grows linearly as all collected data is concatenated. The best-performing CL methods, however, require only an additional 11MB of non-volatile memory to store the samples for rehearsal.

5.6. SUMMARY OF FINDINGS

During inference, CL methods do not impact performance, so the one-second latency is mainly driven by the 1Hz sampling frequency needed to generate a new data window. SmooDER and SmooER may hold the predicted driver ID for up to six seconds, but only during a driver change.

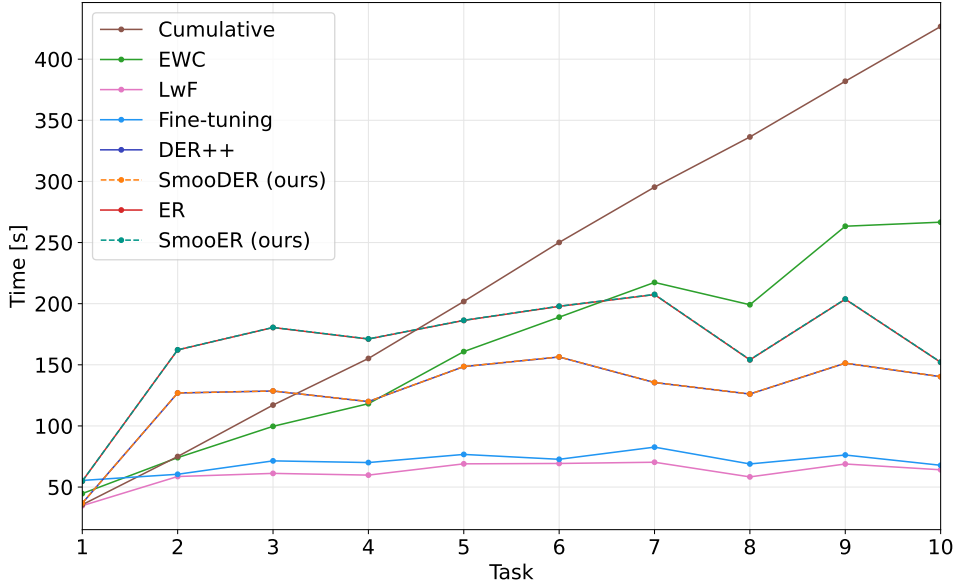


Figure 5.9: Training time per task for each method and baseline in Scenario 3. Cumulative training time increases linearly as more tasks are added since the datasets are concatenated. Fine-tuning and LwF have negligible or no computational overhead. EWC computational time grows since by implementation details each task contributes independently to the regularization. DER++, ER, SmooER, and SmooDER go to saturation when the memory is filled for the first time. The overhead of SmooDER over DER++ is negligible (same for SmooER over ER).

5.6 SUMMARY OF FINDINGS

In this work, we addressed the challenge of Driver Identification. We moved beyond the traditional approach, where all driver data is collected before model deployment, and training occurs offline. In many real-world scenarios, like car-sharing services where new drivers are continuously added as they request to access the vehicle, the traditional complete retraining policy is inadequate due to limited computational resources and memory. On the other hand, training only on new data is impractical because the model would lose the ability to recognize already authorized users, a phenomenon known as catastrophic forgetting. Continual Learning allows the system to not only add new drivers as they come but also to incorporate new data from previously seen drivers. This approach enables a system that progressively adapts and enhances its performance as new data is collected.

The main goal of this work was to apply well-established CL approaches, such as EWC, LwF, ER, and DER++, to static Driver Identification problem and evaluate their effectiveness in reducing Forgetting compared to Fine-Tuning, as well as limiting the resource requirements compared to Cumulative Training when transitioning to the CL formulation. To test our hypothesis, we designed three progressively realistic and complex scenarios. We validate our approach using the OCSLab dataset. Additionally, we took into account the temporal correlation among samples during the inference phase and introduced two new methods, SmooDER and SmooER, which are designed to leverage this information to further minimize Forgetting.

The average accuracy of our baselines across all three scenarios is: 99.40% and 98.52% for the upper bounds (Joint Training and Cumulative Training, respectively) and 18.20% for Fine-Tuning. In contrast, the accuracy for the CL methods is as follows: 88.96% for DER++, 87.93% for ER, 18.56% for LwF, and 18.24% for EWC. These results demonstrate that even in challenging environments like our scenarios, ER and DER++ are effective solutions, as they managed to reduce the linear growth of the required time for each new task of the Cumulative Training approach to an approximately constant time, while only losing an average of about 11% accuracy from Joint Training due to forgetting. Furthermore, our methods, SmooDER and SmooER, proved to be the best approaches, reducing the gap to 2.19% and 2.59%, respectively, with negligible computational overhead compared to DER++ and ER.

5.6.1 DIRECTIONS FOR FUTURE RESEARCH

While our work is extensive in considering multiple aspects of the problem, behavior-based driver identification is a complex field and there are still many opportunities for future research. The proposed approach, although effective is relatively simple. Incorporating an adaptive window size or a dynamic smoothing technique could further minimize misclassifications.

Moreover, there is significant potential for future research into the effects of adversarial attacks and data poisoning on behavior-based driver identification systems that use continual learning. Our system does not introduce any specific vulnerabilities that are easily exploitable. All components are integrated within the microcontroller, and physical access would be required to alter them. However, we assume the system inherits the vulnerabilities of the underlying model. Consequently, adversarial attacks may lead to misclassifications, and data poisoning could degrade model performance over time. Therefore, future work should focus on investigating these aspects and on developing defenses that balance adaptability and robustness.

5.6. SUMMARY OF FINDINGS

Finally, the applicability of the method to high quality data from a HPP remains to be validated to identify potential deployment issues. While this validation focuses on the hydropower domain, the modularity and flexibility of our approach demonstrate potential for future application in other domains with similar properties, such as human activity recognition, healthcare, or predictive maintenance in other industrial sectors.



Feature-Evolving Anomaly Detection

In this thesis, AD is a central theme, explicitly addressed in all chapters except Chapter 5. This is not by chance, as the ability to detect potentially harmful situations with high accuracy while sustaining minimal operational costs is crucial in PdM and quality control. In such contexts, one class (normal operation or good quality products) is naturally well-defined, while the other (faulty operation or defective products) is heterogeneous and often only partially specified. As a result, AD finds applications across a wide range of domains, including fault detection in sensor networks, monitoring of industrial CPS, and fraud detection [241, 242].

Despite recent advances in DL architectures that enable direct AD on complex data such as images, text, and time series by automatically extracting hierarchical features [243], tabular data remains central in many industrial domains [244, 191]. Organizations have historically relied on this format and literature-backed engineered features, making compatible AD methods attractive for their seamless and cost-effective integration into existing systems.

As anticipated in Section 2.5.3, a major challenge in deploying AD systems lies in their ability to adapt to evolving environments. In this chapter, we focus on Feature-Evolving (FeV), a scenario where new features are introduced over time [14]. This situation frequently arises in practice, as industrial systems are upgraded with additional sensors or data sources, resulting in an incremental increase in input dimensionality. While data distribution shifts have been widely studied [245, 246, 247, 248], few works have explored incorporating new features into already trained models. Traditional AD models are designed for fixed feature spaces, and ignoring new features risks leaving anomalies undetected, while retraining from scratch is often impractical, particularly when historical data is unavailable due to privacy, storage, or cost constraints.

6.1. PROXY TASK

To address this gap, we propose Root-forced Isolation Forest (RootIF), a novel FeV AD approach based on Isolation Forest. Our method integrates new features into existing models using a few additional samples, which enables faster deployment and reduces the system’s vulnerability window. At the same time, it preserves the knowledge accumulated by the original model without requiring access to historical training data, thereby strengthening data privacy. The core innovation lies in augmenting the original forest with new trees that are forced to split on the newly introduced features. At evaluation, we introduce an adaptive unsupervised weighting mechanism that balances the contributions of original and new features. In particular, we leverage Depth-based Feature Importance of Isolation Forest (DIFI), a local feature importance method tailored for iForest, to estimate the contribution of each new feature to point isolation and use these contributions to weight both original and newly added trees.

Our contributions are summarized as follows:

- We introduce a novel framework for FeV AD that integrates new features into an existing iForest model with minimal new data and no access to historical data.
- We propose an adaptive, unsupervised weighting mechanism for anomaly scoring that balances the contributions of original and newly added features.
- We conduct extensive experiments on multiple real-world datasets, demonstrating the effectiveness and efficiency of our approach compared to baseline and SOTA methods.

The remainder of this chapter is organized as follows. Section 6.1 introduces the proxy task that relates this work to hydroelectric PdM. Section 6.2 provides the necessary background and reviews related work, with a focus on SOTA approaches for FeV. Section 6.3 presents the foundational algorithms, the simplified problem definition derived from Section 3.2, and the proposed RootIF method. Section 6.4 describes the experimental setup, while Section 6.5 presents and discusses the results. Finally, Section 6.6 summarizes the contributions and outlines directions for future research.

6.1 PROXY TASK

In this work, we propose and evaluate a FeV AD method based on iForest. We simulate the scenario of features being introduced over time by initially training the model on a subset of features and progressively adding new features. To assess our method, we use the ODDS collection of AD datasets [165] as proxy tasks.

A central focus of our work is efficiency. Our approach enables an existing model to detect faults in the new feature space using only a small number of new samples for the update. This allows the model to preserve accumulated knowledge without requiring access to historical data and supports a trade-off between asymptotic performance and computational cost.

In the context of HPP fault detection, this efficiency is especially relevant. If a company invests in installing a new sensor or in developing a pipeline to extract a new feature, it is likely that this new information is essential for recognizing faults that could not be detected before. Our approach makes it possible to quickly adapt the model to the new information without retraining from scratch or storing historical data, reducing the time during which the system remains vulnerable.

The method can be applied both on the edge and in the cloud. At a low level, it can monitor raw sensor data, either directly for protection mechanisms or as an intermediate step to decide which data to send to the cloud. At a higher level, it can work with more complex features extracted from raw data in the cloud, to detect anomalies and alert human operators.

6.2 BACKGROUND AND RELATED WORK

Dynamic environments are common across many domains, including industry, and they pose a significant challenge when applying ML solutions. Without specific approaches to handle such changes, model performance can quickly degrade in deployment. Many approaches have been proposed to address these issues, and in Chapter 5 we explored how CL techniques can help DL models adapt to shifting data distributions or the introduction of new classes. What has received far less attention, however, is the challenge posed by an evolving feature space.

There is no single, universally accepted term for this problem. Different works adopt slightly different definitions and labels, with terms such as dynamic feature spaces [249] and feature streaming [250] commonly used. In this thesis, we adopt the definition of Feature-Evolving (FeV) [14], as it most closely reflects our scenario.

In practice, FeV rarely occurs in isolation. It is usually intertwined with other types of distributional shifts. With this in mind, the following sections first review the broader AD literature on dynamic environments, then narrow the focus on FeV. We conclude by examining xStream [251], a SOTA AD model designed to address online learning in the presence of both distributional shifts and evolving feature spaces. It will be the main comparison point for our proposed RootIF.

6.2.1 DISTRIBUTIONAL SHIFTS

Most of the literature on dynamic environments focuses on the main challenges that, according to the taxonomy of Section 2.5.3, can be defined as *covariate shift*, *label shift*, and *concept drift*.

In [252], the authors proposed a DL approach that minimizes a regularization term inspired by causal inference. This allows the model to learn latent representations that are invariant to changes in the input distribution, thus mitigating distribution shifts which could otherwise be mistaken for anomalies. Traditional ML models like One-Class Support Vector Machine have also been adapted to handle distributional shifts. In [247], the authors proposed an adaptive version of the model that incrementally tunes the kernel parameters to consider new data points in the normal distribution, thus adapting to gradual changes.

iForest has also been extended to streaming, non-stationary settings. In [245], the authors introduced a variant that monitors concept drift using nearest-neighbor density divergence and dynamically updates its tree ensemble. Trees are added or removed in proportion to the detected drift, helping to reduce false positives. Similarly, [253] proposed a hybrid drift detector that combines statistical tests with performance-based evaluation of tree path lengths in a sliding-window framework. Trees that degrade due to drift are discarded, ensuring the ensemble remains representative of the current distribution.

Although our proposed approach is fundamentally different in both task and mechanism, these last two works offer interesting insights into how iForest can be adapted to an online learning setting. This remains the most promising direction for improving our work, as we will restate in Section 6.6.1.

6.2.2 EVOLVING FEATURE SPACE

The challenge of evolving feature spaces has two main dimensions. First, individual features may become intermittently or permanently unavailable due to sensor faults, communication problems, or privacy constraints. This leads to missing values in incoming samples, which are typically addressed via imputation by inferring values based on the observed distributions [254]. The second dimension, which is the focus of this work, is the introduction of entirely new features that were not present during training. This poses a more complex problem, as the model must adapt to an expanded feature space without prior knowledge of the new dimensions.

A well-known traditional online learning AD model that natively addresses this issue is xStream, which we will describe in detail later in Section 6.2.3. Broadening the scope to DL

models, [255] proposes a stochastic recurrent deep neural network that combines a LSTM encoder with probabilistic quantile regression, providing principled uncertainty estimates through a recurrent quantile loss. Their method learns latent temporal embeddings that are not tied to a fixed feature set, together with an incremental update mechanism that adjusts thresholds and model parameters as new features appear. This design avoids retraining from scratch and maintains robustness under heterogeneous, high-dimensional, and evolving conditions.

Although such deep recurrent models achieve high accuracy in streaming, FeV settings, they are computationally demanding, require specialized hardware, and are most suitable when strong temporal dependencies are present. In many practical applications, however, data arrives in batches rather than continuous streams, temporal correlations are weak or irrelevant, and efficiency is essential. In these cases, iForest offers an attractive alternative: it is lightweight, scalable to high-dimensional data, and does not rely on distributional assumptions. Extending iForest to handle newly introduced features allows us to preserve these benefits while enabling AD in FeV spaces under non-streaming conditions, providing a simple and efficient solution where deep sequential models would be unnecessary or impractical.

6.2.3 xSTREAM

This model, introduced in [251], is a SOTA approach for AD in online, streaming and FeV environments. It is particularly relevant to our work as it directly addresses our challenges and serves as a primary benchmark for evaluating our proposed RootIF.

xStream essentially combines three key ideas: random projections, ensemble learning, and density estimation through recursive partitioning. It builds on the concept of random projections, using K hash tables to reliably project a sample with an arbitrary number of features into a fixed K -dimensional space.

$$y[i] = \sum_{j=1}^d h_i(f_j) \cdot x[j] \quad \text{for } i = 1, \dots, K$$

Where x is the original d -dimensional input vector, y is the projected K -dimensional vector, and $h_i(f_j)$ is a hash function that maps the feature with $id = f_j$ of x to the i -th dimension of y .

The ensemble of M Half-Space Chains of fixed depth D (with D levels) is then used to divide the projected space into bins and *count-min-sketch* are used to efficiently estimate local

6.2. BACKGROUND AND RELATED WORK

densities in those bins (each *count-min-sketch* uses m hash tables to estimate the density). Essentially at each level a random dimension is selected and partitioned into bins, with the bin width halving at each deeper level if the same dimension is selected again. This creates a multiscale discretization of the space, allowing the model to capture local density variations at different granularities.

Finally, the anomaly score for a point is computed as the average across all chains of the minimum normalized bin count across all levels, with lower counts indicating higher anomaly likelihood.

UPDATE

The model is designed to allow online modification of the values of existing features, as well as the addition of new features. This is handled by caching the N most recently used points. In case of an update the point is removed from the cache, projected to remove the effect on the chains, inserted as the cache's head with its updated value, and then re-projected to update the chains bins.

To adapt to changes in the data distribution, a sliding window approach is used. Two versions of the model are maintained, one trained on the previous window of size ϕ which is used to compute anomaly scores and one on the current window that is only updated with the iterative approach discussed above. At the end of each window, the current model replaces the previous one, and a new current model is initialized in $O(1)$ time.

COMPUTATIONAL COMPLEXITY

The complexity is presented in the original paper [251] on a per-update basis, where an update is a triplet of (*point_id*, *feature_id*, *value*) which can add a new point, append a new feature to an existing one or modify a feature of a previously learned point. The update and score complexity are the same as the procedure only differs in reading or updating the *count-min-sketch* tables.

A formulation compatible with our experiments can be obtained by considering a batch of S points with same number of features F ($x \in \mathbb{R}^F$) as $S \cdot F$ updates, one for each feature from each sample. Note that while in our experiment each batch is composed of the same points with one less hidden feature (see Section 6.4), in a real-world scenario the points in a batch can be completely different and thus one update per sample would not be representative for the scenario.

The reported time complexity is $O(K \cdot m \cdot D \cdot M)$, as for each of the M chains, D levels are traversed and the relative *count-min-sketch* hash tables are updated or queried, each

requiring $O(K \cdot m)$ operations, mainly driven by the computation of one hash function on the bin-index vector of size K for each of the m hash tables. This results in an overall complexity of $O(K \cdot m \cdot D \cdot M)$ per update, which for a set of S points becomes $O(S \cdot F \cdot K \cdot m \cdot D \cdot M)$.

6.3 METHODOLOGY

In this section we introduce the proposed RootIF algorithm, an extension of the classical iForest designed for the FeV setting. We first review DIFFI, a local feature importance method for iForests, which we exploit to adaptively balance the contributions of historical and new trees. Next, we formalize the problem and describe the main components of our approach: the update procedure for integrating new features and the adaptive anomaly score computation. We conclude with an analysis of the computational complexity of the proposed method.

6.3.1 DIFFI: DEPTH-BASED FEATURE IMPORTANCE OF ISOLATION FORESTS

Depth-based Feature Importance of Isolation Forest (DIFFI) is a post-hoc interpretability algorithm specifically designed to compute feature importance scores for trained iForest models [189]. DIFFI has two variants: a global version, which estimates importance across the entire training dataset, and a local version, which quantifies the contribution of each feature to the anomaly score of a specific instance. In this work, we focus exclusively on the local variant.

Given a data point $x \in \mathbb{R}^p$ predicted as anomalous, let $\text{Path}(x, t)$ denote the sequence of internal nodes traversed by x in the t -th isolation tree, from root to leaf. Each internal node $v \in \text{Path}(x, t)$ is associated with a splitting feature $j \in \{1, \dots, p\}$. Let $\tilde{d}_t(x)$ denote the depth (or path length) of x in tree t , and let $d_{\max}$ be the maximum tree depth. The importance score I_j is accumulated for each feature j by iterating over all trees and nodes. Whenever a node $v \in \text{Path}(x, t)$ splits on feature j , I_j is incremented by:

$$\Delta = \frac{1}{\tilde{d}_t(x)} - \frac{1}{d_{\max}}. \quad (6.1)$$

The correction term ensures that features are not credited when x is not effectively isolated (i.e., when $\tilde{d}_t(x) = d_{\max}$). Without it, all features along the path would receive non-zero attributions even if the tree failed to isolate the instance.

In parallel, a counter C_j tracks how many times feature j appears in the paths $\text{Path}(x, t)$

6.3. METHODOLOGY

across the forest. The final local feature importance is then defined as:

$$LFI(x)_j = \frac{I_j}{C_j}, \quad \forall j \in \{1, \dots, p\}. \quad (6.2)$$

The underlying intuition is that features contributing to shorter isolation paths are more informative for detecting anomalies. Normalization by C_j ensures that importance scores remain invariant to how frequently a feature is randomly selected across the forest.

6.3.2 PROBLEM FORMULATION

Let $X \in \mathbb{R}^{n \times m}$ denote a dataset with n samples and m features, and let $\mathcal{F}_X$ be an iForest model trained on X . Suppose a new dataset $X' \in \mathbb{R}^{n' \times m'}$ becomes available, where $m' = m + k$ for some $k > 0$ and possibly $n' \ll n$. The new input matrix X' contains the original m features, augmented with k additional ones. We assume that the first m features in X' correspond exactly (in both order and meaning) to those in X , and that no distribution shift occurs in these features.

The goal is to compute anomaly scores for new samples $x' \in \mathbb{R}^{m+k}$ in the augmented feature space, without discarding $\mathcal{F}_X$ and retraining a new model solely on X' . While full retraining is always an option, it presents two major drawbacks:

1. When n' is small, the resulting forest lacks statistical robustness.
2. The knowledge embedded in the large-scale historical model $\mathcal{F}_X$ is discarded.

Our objective is therefore to design a mechanism that preserves the stability and information captured by $\mathcal{F}_X$, while effectively incorporating the new features contained in X' .

6.3.3 ROOT-FORCED ISOLATION FOREST

To extend the iForest model to the FeV setting, we propose RootIF, whose objective is to preserve the stability of an already trained model while efficiently adapting to the augmented feature space. This is particularly relevant when access to historical data is restricted, for example due to privacy or storage constraints. Our method leverages a small amount of new data to build additional trees that explicitly incorporate the new features, thereby reducing computational burden, enabling fast deployment, and minimizing the system's vulnerability window. The final model combines the predictive power of the retained historical trees with that of the newly trained ones through an adaptive, instance-specific aggregation mechanism.

The framework consists of three key ideas: (i) construction of new trees on the dataset $X' \in \mathbb{R}^{n' \times (m+k)}$, where the initial splits are forced on the newly introduced features; (ii) merging of new and old trees from the original forest $\mathcal{F}_X$, with the possibility of pruning to control the final forest size; and (iii) adaptive score aggregation based on DIFFI local feature importance, which balances the contributions of historical and new trees depending on the relevance of their split features to each sample.

This formulation yields a hybrid iForest that preserves the knowledge encoded in the historical model while integrating the discriminatory power of new features, even under limited availability of new data.

UPDATE PROCEDURE

Let $\mathcal{F}_X = \{t_1, t_2, \dots, t_T\}$ denote the original iForest trained on X . To incorporate the new features, we construct a set of T' additional trees $\mathcal{F}_{X'} = \{t'_1, t'_2, \dots, t'_{T'}\}$ using the new dataset $X' \in \mathbb{R}^{n' \times (m+k)}$. As described in Algorithm 1, the key modification is that the root node of each new tree is forced to split on one of the k newly added features (randomly selected among them), while all subsequent splits follow the standard random procedure of iTrees. This guarantees that every new tree is sensitive to the additional features, while maintaining randomness and diversity deeper in the structure.

The updated ensemble is obtained by combining the original and new trees: $\mathcal{F} = \mathcal{F}_X \cup \mathcal{F}_{X'}$. Since the complexity of training a forest is $O(T \cdot \psi \log \psi)$, where ψ is the subsampling size (see Section 4.1.1), updating only requires training the T' new trees at cost $O(T' \cdot \psi \log \psi)$. At inference, however, the complexity grows linearly with the forest size, $O((T + T') \cdot \log \psi)$ per sample, which can become impractical if the ensemble grows indefinitely. To address this, we introduce a fixed forest size $\bar{T}$, leaving the choice of $\bar{T}$ and T' to the user depending on the application's accuracy-efficiency trade-offs. Training fewer trees results in faster updates, while if enough data is available training more trees provides better adaptation to the new features. The final inference cost scales with $\bar{T}$.

With this constraint, when updating the model we consider two scenarios:

- If $T + T' \leq \bar{T}$ (or no limit is imposed $\bar{T} = \infty$), the two forests are simply merged.
- If $T + T' > \bar{T}$, pruning is required. We assign weights

$$w_{t'} = T \quad \forall t' \in \mathcal{F}_{X'}, \quad w_t = T' \quad \forall t \in \mathcal{F}_X,$$

and sample without replacement $\bar{T}$ trees from $\mathcal{F}_X \cup \mathcal{F}_{X'}$ with probability proportional to these weights. This scheme samples new trees proportional to the number of old

6.3. METHODOLOGY

trees and samples old trees proportional to the number of new trees, which tends to retain all new trees when few are added and retains historical trees when many new ones are added.

Algorithm 1 Root-forced Isolation Tree Construction

Input: Input data $X' \in \mathbb{R}^{n' \times (m+1)}$, Current depth d , Maximum depth $d_{\max}$

Output: Root-Forced Isolation Tree Node

```

if  $d \geq d_{\max}$  or  $|X'| \leq 1$  then
    return externalNode {size  $\leftarrow |X'|$ }
end if
if  $d = 0$  then                                ▶ Root is split on the last (new) feature
    Let  $j \leftarrow m + 1$                             ▶ Index of the added feature
    Let  $min \leftarrow \min(X'[:, j]), max \leftarrow \max(X'[:, j])$ 
    Randomly select split point  $p \in (min, max)$ 
else
    Let  $Q \leftarrow \{1, 2, \dots, m + 1\}$                 ▶ All attribute indices
    Randomly select attribute  $q \in Q$ 
    Let  $min \leftarrow \min(X'[:, q]), max \leftarrow \max(X'[:, q])$ 
    Randomly select split point  $p \in (min, max)$ 
end if
 $X'_l \leftarrow \{x \in X' \mid x[q] \leq p\}$ 
 $X'_r \leftarrow \{x \in X' \mid x[q] > p\}$ 
 $left \leftarrow \text{ConstructTree}(X'_l, d + 1, d_{\max})$ 
 $right \leftarrow \text{ConstructTree}(X'_r, d + 1, d_{\max})$ 
return internalNode {splitAtt  $\leftarrow q$ , splitValue  $\leftarrow p$ , left  $\leftarrow left$ , right  $\leftarrow right$ }

```

ADAPTIVE ANOMALY SCORE

In Section 6.3.3 we allowed the user to adapt the update behavior to the specific application context by selecting the number of new trees T' and the maximum forest size $\bar{T}$. These choices directly determine the final ensemble composition since the new trees which can exploit the additional features can be underrepresented. To make the model behavior robust across different values of T' and $\bar{T}$, we introduce an adaptive anomaly score (Algorithm 2) that, for each instance, dynamically weights the contribution of each tree according to the relevance of the features it can see.

Starting from the standard iForest anomaly score of Equation 4.2, $s(x, \psi) = 2^{-\frac{E(h(x))}{c(\psi)}}$, we modify the expected path length $E(h(x))$ into a weighted average that balances historical and new trees. Let $\mathcal{F} = \mathcal{T} \cup \mathcal{T}'$ denote the updated ensemble, where $\mathcal{T}$ are the retained original

trees trained on X and $\mathcal{T}'$ the added new trees trained on X' . The weighted path length is defined as:

$$\bar{h}(x) \doteq \frac{\sum_{t \in \mathcal{T}} w_t(x) h_t(\tilde{x}) + \sum_{t \in \mathcal{T}'} w_t(x) h_t(x)}{\sum_{t \in \mathcal{T}} w_t(x) + \sum_{t \in \mathcal{T}'} w_t(x)}, \quad (6.3)$$

where $\tilde{x} \in \mathbb{R}^m$ is x with the new features removed, $h_t(x)$ is the path length in tree t , and $w_t(x)$ is an instance-dependent weight.

The weights $w_t(x)$ are derived from local feature importance computed by DIFFI (Section 6.3.1). Let $LFI_j(x)$ denote the importance of feature j for sample x . Define the set of trees that can split on each feature j as:

$$\mathcal{T}_j \doteq \{t \in \mathcal{T} \cup \mathcal{T}' \mid \text{tree } t \text{ contains feature } j\},$$

with cardinality $|\mathcal{T}_j|$. We distribute the importance of feature j proportionally among its trees, and obtain the weight of tree t as:

$$w_t(x) \doteq \sum_{j: t \in \mathcal{T}_j} \frac{LFI_j(x)}{|\mathcal{T}_j|}. \quad (6.4)$$

If the new features are irrelevant for an instance, $LFI_j(x)$ values are small and the weights approximate uniform averaging, thereby recovering the behavior of a standard iForest. If the new features are relevant and new trees isolate the instance early, those trees accumulate larger $w_t(x)$, amplifying their contribution to the anomaly score.

6.3.4 COMPUTATIONAL COMPLEXITY

We now analyze the computational complexity of our method. As in the standard practice, we report training and update costs with respect to the training dataset size n , while inference costs are given per sample. This separation reflects the different operational settings: training and updating are performed once taking into account the whole training dataset, while inference is performed independently for each sample and repeated at scale during deployment.

UPDATE COMPLEXITY

The cost of building the initial iForest is $O(T \cdot \psi \log \psi)$, where T is the number of trees and ψ is the subsampling size. Training T' new trees adds an update cost of $O(T' \cdot \psi \log \psi)$ since each new tree is trained independently on subsamples of the updated dataset. The

Algorithm 2 Adaptive anomaly score computation**Input:** Sample $x \in \mathbb{R}^{m+1}$, Old trees $\mathcal{T}$, New trees $\mathcal{T}'$ **Output:** Anomaly score $a(x) \in \mathbb{R}$

- 1: Initialize: $S_f \leftarrow 0, C_f \leftarrow 0 \quad \forall f \in \{1, \dots, p\}$
- 2: **for all** tree t in $\mathcal{T} \cup \mathcal{T}'$ **do**
- 3: Obtain decision path features for x : $\text{Path}(x, t) = [f_1, f_2, \dots, f_{k_t}]$
- 4: $\triangleright$ ordered list of features in path
- 5: Obtain leaf corrected depth $d_t(x)$ where x ends in tree t
- 6: **for all** features f in $\text{Path}(x, t)$ **do**
- 7: Accumulate importance score: $S_f \leftarrow S_f + \frac{1}{d_t(x)}$
- 8: Increment counter: $C_f \leftarrow C_f + 1$
- 9: **end for**
- 10: **end for**
- 11: Normalize per feature: $\tilde{S}_f \leftarrow \frac{S_f}{C_f}, \quad \text{if } C_f > 0 \quad \text{else} \quad \tilde{S}_f \leftarrow 0$
- 12: Normalize scores to sum to 1: $\hat{S}_f \leftarrow \tilde{S}_f / \sum_f \tilde{S}_f$
- 13: Let N_f be the number of trees in $\mathcal{T} \cup \mathcal{T}'$ that split on feature $f, \forall f$
- 14: For each tree t , compute weight:

$$w_t \leftarrow \sum_{f \in \text{Path}(x, t)} \frac{\hat{S}_f}{N_f}$$

- 15: Normalize tree weights to sum to 1: $\hat{w}_t \leftarrow w_t / \sum_t w_t$
- 16: Compute weighted average depth:

$$\bar{d}_x \leftarrow \sum_t \hat{w}_t \cdot d_t(x)$$

- 17: Compute anomaly score:

$$a(x) \leftarrow 2^{-\frac{\bar{d}_x}{c(\psi)}}$$

root-forcing step has negligible $O(1)$ cost. Accounting also for the merging step, where $\bar{T}$ trees are sampled from the union of old and new trees, we obtain an additional $O(T + T')$ term. The overall update complexity is therefore

$$O(T' \cdot \psi \log \psi + T + \mathcal{P}'),$$

Where the $\mathcal{P}'$ highlights that the linear contribution of T' is dominated by the training term $O(T' \cdot \psi \log \psi)$.

If the size of the forest is limited $T \leq \bar{T}$ then the update complexity remains in line with the base iForest and is for the large part driven by T' and $\bar{T}$, user-defined parameters.

INFERENCE COMPLEXITY

At inference time, the baseline cost of traversing all trees in the updated forest is $O((T + T') \cdot \log \psi)$. When adaptive weighting via DIFFI is applied, an additional feature-importance computation is required. DIFFI computes local feature contributions for each sample by inspecting its traversal path through each tree, which costs $O((T + T') \cdot \log \psi)$. Distributing these importances to the corresponding trees introduces a further $O((T + T') \cdot F)$, where F is the total number of features.

Thus, the overall inference complexity with DIFFI weighting is

$$O((T + T') \cdot (\log \psi + F)),$$

6.4 EXPERIMENTAL SETUP

To evaluate the effectiveness of RootIF, we conduct experiments on 17 real-world anomaly detection datasets from the ODDS repository [165]. The objectives of our evaluation are threefold: (i) to verify the method’s ability to retain knowledge from the original model, (ii) to evaluate how effectively it integrates newly introduced features, and (iii) to analyze how performance varies with different amounts of new data available.

In the remainder of this section, we describe the comparative baselines, the datasets and data preprocessing steps, the experimental procedure, the evaluation metrics, an ablation study assessing the contribution of each model component, and finally, the implementation details.

6.4.1 BASELINE METHODS

We evaluate our proposed approach against both classical and SOTA baselines:

- **from_scratch:** A stateless iForest that is re-initialized and retrained at each training-test iteration. This represents the common strategy of adapting to new features by discarding the past model and learning a new one from scratch.
- **fixed:** An iForest trained only during the first *experience* and then kept fixed, with no further updates. This baseline quantifies the performance drop when the model ignores new features, providing a lower-bound reference.
- **xStream:** The online anomaly detection algorithm [251], which is widely recognized as a SOTA streaming baseline. It is specifically designed to handle evolving feature spaces

and serves as a strong benchmark for our method (see Section 6.2 for more details).

6.4.2 DATASETS

All experiments were conducted on 17 real-world datasets from the ODDS (Outlier Detection Datasets) repository [165]:

- **Annthyroid:** Medical dataset for detecting thyroid disorders. Normal function is treated as inliers, while hyper and hypothyroidism are considered outliers. Includes six numerical attributes from thyroid-related tests.
- **Breastw:** Breast cancer diagnostic data distinguishing benign from malignant tumors. Malignant cases are outliers; benign cases are inliers. Contains nine ordinal features ranging from 1 to 10.
- **Cardio:** Cardiovascular records for detecting abnormal conditions. Pathological cases are labeled as outliers. Includes 21 numerical health-related attributes such as blood pressure and heart rate.
- **Ionosphere:** Radar signals from the ionosphere classified as “good” or “bad” returns. Bad returns are treated as outliers. Contains 33 numerical signal features.
- **Letter:** Handwritten character images reduced to 32 numerical features. Deviations from the majority class are treated as outliers.
- **Mammography:** Image and clinical features extracted from mammograms. Anomalous samples correspond to rare cancer-indicative cases. Includes six numerical attributes.
- **MNIST:** Subset of the MNIST handwritten digit dataset. Digits outside the reference class are treated as outliers. Features correspond to pixel intensity values.
- **Optdigits:** Handwritten digit recognition dataset. Digits 1-9 are inliers; digit 0 is an outlier. Contains 64 numerical pixel intensity features from 8x8 grayscale images.
- **Pendigits:** Pen stroke recordings of handwritten digits. Abnormal formations are outliers. Includes 16 numerical (x, y) coordinate features.
- **Pima:** Medical dataset of Pima Indian women. Anomalies correspond to diabetes-related irregularities. Includes eight demographic and medical features.
- **Satellite:** Multispectral satellite imagery. Anomalous regions indicate unusual terrain or misclassification. Includes 36 reflectance-based features.
- **Satimage-2:** Variant of the Satellite dataset where one class is treated as normal and

others as outliers. Includes 36 spectral features from 3x3 pixel windows.

- **Thyroid:** Medical dataset identifying thyroid dysfunction. Hyperfunction samples are outliers; normal and subnormal are inliers. Includes six blood test features.
- **Vertebral:** Spine X-ray biomechanical data. Abnormal configurations are inliers, while normal cases are treated as outliers. Contains six numerical attributes.
- **Vowels:** Phoneme recognition dataset based on vowel sounds. Mispronunciations or unusual pronunciations are outliers. Includes 12 acoustic features.
- **Wbc:** Wisconsin Breast Cancer dataset distinguishing benign from malignant cells. Malignant cases are outliers; benign cases are inliers. Contains 30 cytological features.
- **Wine:** Chemical analysis of wines from three cultivars. Class 1 is treated as composed of outliers; Classes 2 and 3 as inliers. Includes 13 chemical features.

Key dataset statistics, including number of samples, dimensionality, and contamination rate, are reported in Table 6.1.

Dataset ID	Num samples	Num features	Contamination
annthyroid	7200	6	7.42%
breastw	683	9	35%
cardio	1831	21	9.6%
ionosphere	351	33	36%
letter	1600	32	6.25%
mammography	11183	6	2.3%
mnist	7603	100	9.2%
optdigits	5216	64	3%
pendigits	6870	16	2.2%
pima	768	8	34.8%
satellite	6435	36	32%
satimage-2	5803	36	1.2%
thyroid	3772	6	2.5%
vertebral	240	6	12.5%
vowels	1456	12	3.4%
wbc	378	30	5.5%
wine	129	13	7.7%

Table 6.1: Statistics of ODDS datasets

6.4. EXPERIMENTAL SETUP

DATA PREPARATION

ODDS is a collection of well-known datasets that have been preprocessed and specifically adapted for AD tasks. Consequently, our preprocessing pipeline was limited to splitting the data, using a *test_ratio* of 20%, in order to evaluate the true generalization performance of the models. Performing this split as the very first step ensures that no information leakage occurs, i.e., that test data is never used during training.

Although iForest-like models are invariant to feature scaling, we applied mean-variance normalization (already discussed in Section 5.4.1) to each feature to guarantee a fair comparison across all methods. Normalization statistics were computed exclusively on the training set and later applied to the test set, preventing any data leakage.

Finally, prior work has shown that increasing model complexity through hyperparameters provides saturating benefits [251]. For this reason, we adopted the standard values commonly used in the literature for iForest-based models and selected the saturation-level values reported in the original xStream paper, thereby avoiding costly hyperparameter searches. This choice is particularly advantageous in our setting, as reserving a separate validation set would be problematic for the smaller datasets listed in Table 6.1. The only hyperparameter that would normally require tuning is the threshold used to convert anomaly scores into binary predictions. Since our main evaluation metric is Average Precision (AP) (see Section 6.4.4), which is threshold-independent, this does not affect the reported results.

6.4.3 EXPERIMENTAL PROCEDURE

The objective of our experimental procedure is to evaluate the model’s ability to adapt to new features, assess the sample efficiency required to reach acceptable performance, and verify that the accumulated knowledge is properly exploited.

To quantify these aspects, we employ two primary metrics, which are detailed in Section 6.4.4. We use AP to assess absolute detection quality; high AP values indicate successful adaptation to the new features. Complementing this, we use Resource-Weighted Performance (RWP) to evaluate the adaptation process as a whole by measuring the area under the AP curve as a function of the update dataset size. RWP creates a unified metric where the score is penalized in three scenarios: (i) if the model fails to reach a high convergence AP (reducing the total height), (ii) if the model fails to retain accumulated knowledge (causing an initial drop), or (iii) if the model requires many samples to adapt (delaying the rise in performance).

Moreover, it is worth to note that in a real-world scenario, the number of samples reported in Section 6.5 corresponds to the number of new states of the system that must be observed

in order to update the model and achieve the reported performance. This aspect is crucial, as one hundred samples may be collected in milliseconds in high-frequency systems or in seconds if using window-based features. Nevertheless, such samples often represent similar system conditions and therefore provide limited statistical robustness when compared to the same number of, for instance, thyroid analyses from different patients.

To ensure clarity, the experimental settings are summarized in Table 6.2 before being explained in detail.

Setting Name	Value / Description
test_ratio	0.2
n_seeds_exp	50
n_seeds_shap	30
n_experiences	3
n_blocks_per_experience	13
n_start_trees	100
n_trees_progression	[1, 2, 5, 10, 25, 50, 100]
xStream_K	100
xStream_M (parameter c)	100
xStream_D (parameter d)	25

Table 6.2: Settings used in our experiments

EXPERIMENT

Our experiment is composed of an inner and outer loop, which are executed for a total of n_seeds times, each with a different random seed to ensure statistical robustness.

The outer cycle corresponds to the task of FeV learning: starting from $F - n_experiences$ features, we progressively add one feature at each iteration. The inner cycle explores the effect of the number of instances available for training on performance. Here, the training data is partitioned into n_blocks subsets. At iteration b , all subsets with index $i \leq b$ are concatenated and used for training, until all training data is included. The subset sizes grow exponentially, reflecting the fact that data additions to a small dataset yield disproportionately larger performance gains than equivalent additions to a large dataset.

Within each iteration of the inner cycle, models are trained and evaluated. All metrics presented in Section 6.4.4 are computed on the test data, following common practice [256].

From a practical standpoint, adding a new sensor represents a cost that is only justified if it improves fault detection capabilities. Operating under the assumption that sensors

6.4. EXPERIMENTAL SETUP

are **introduced purposefully to capture missing information**, we avoid wasting limited *experiences* on noisy inputs by ranking features according to their global importance and observing the impact of adding the most important ones.

Feature importance rankings are computed with TreeSHAP [257], a model-specific interpretability method for ensembles of decision trees. We deliberately avoid using DIFFI for this task to prevent introducing a double influence of DIFFI into the experiments. While DIFFI is preferred in our models due to its speed and negligible overhead during inference, TreeSHAP is acceptable here since the ranking is computed only once per dataset.

Model training follows these rules:

- The model state is saved only at the end of each *experience*, after processing the complete training dataset. Intermediate *blocks* are processed in a memoryless fashion.
- The *fixed* baseline stops training after the first *experience*, representing a static model that is never updated.
- The *from_scratch* baseline is reinitialized and retrained at every *block*, representing a model that cannot access historical knowledge.

Finally, to assess how the number of new trees trained in the update influences the model, we repeated the experiments varying this parameter according to Table 6.2, while keeping random seeds fixed.

6.4.4 EVALUATION METRICS

Evaluating AD models is not as straightforward as in standard classification tasks, since commonly used metrics like accuracy may provide a misleading sense of performance. This is mainly due to two specific challenges.

First, AD datasets are inherently imbalanced. Anomalies are rare by definition, often making up only a small fraction of the data. In such cases, a model that simply predicts all points as *normal* can still achieve high accuracy despite failing to detect any anomalies. As a result, accuracy offers little insight into actual model performance. Metrics that emphasize the minority class, such as precision, recall, and F1-score, are more informative because they directly reflect how well the model identifies rare anomalies.

Second, most AD models do not directly predict whether a point is *normal* or *anomalous*. Instead, they attribute to each instance a score that reflects how unusual it is. A classification threshold must then be set to distinguish between the two classes. This threshold is typically optimized on validation data or application-specific constraints. Its selection involves an

important trade-off: lowering the threshold increases recall but also raises the number of false positives; whereas raising it reduces false alarms but risks missing true anomalies. Because metrics like the F1-score depend entirely on this threshold, they capture model performance only at a single operating point, rather than reflecting its behavior across all thresholds.

AVERAGE PRECISION

To address both the extreme class imbalance and the threshold-dependency issue, we have chosen AP as our primary evaluation metric. AP bypasses the need to select a single threshold by summarizing the precision-recall curve across all possible operating points. This provides a single, comprehensive score that reflects the model's ability to consistently rank anomalies higher than normal data, which is the ultimate goal of a detection algorithm.

Formally, as defined in Equation 6.5, AP is the area under the precision-recall curve, calculated as a weighted sum of precisions at each threshold, with the change in recall serving as the weight.

$$AP = \sum_n (R_n - R_{n-1}) P_n \quad (6.5)$$

Here, P_n and R_n are the precision and recall, respectively, at the n -th threshold in the ranked list of scores. In essence, AP rewards models that maintain high precision as recall increases, providing a robust measure of performance that is well suited to the challenges of AD.

RESOURCE-WEIGHTED PERFORMANCE

This choice for the evaluation metrics allow us to compare model performance independently of the threshold choice, while also accounting for dataset imbalance. Additionally, the design of the experiments allow us to assess how the amount of available training data affects performance. In this setting, selecting the best model becomes a multi-objective optimization problem, as both performance and resource consumption must be considered. To reduce this to a single-objective problem, we assign appropriate weights to combine the two aspects into a single performance metric.

In general, in ML a performance metric can be viewed as a sequence $(m_n)_{n \in \mathbb{N}}$ that is bounded,

$$D \leq m_n \leq U \quad \forall n \in \mathbb{N},$$

and converges to some finite limit $m_n \rightarrow L < \infty$ as $n \rightarrow \infty$, either by design or in practice. The boundedness is straightforward: for example, AP is bounded in $[0, 1]$ by definition, while MSE is effectively bounded by the maximum representable value on a computer. Ensuring

6.4. EXPERIMENTAL SETUP

convergence, however, requires restricting the analysis to models that actually learn the task, meaning their loss function converges to an optimum.

These properties allow us to define our metric as a weighted average of the scores:

$$A_T = \frac{\sum_{r=0}^T w_r m_r}{\sum_{r=0}^T w_r} \quad (6.6)$$

Here, $(m_n)_{n \in \mathbb{N}}$ represents the performance scores, and $(w_n)_{n \in \mathbb{N}}$ is chosen to reflect the resources required to achieve them. Assuming $w_n \geq 0 \forall n \in \mathbb{N}$, as it is reasonable to assume that achieving results entails consuming resources, two main cases arise depending on the behavior of

$$W_T = \sum_{n=0}^T w_n.$$

Case 1: $W_T \rightarrow \infty$ as $T \rightarrow \infty$. This corresponds to a *regular summability method*, where a lower triangular matrix $a_{ij} = w_j/W_i$ transforms one sequence into another with the same limit. In this case, $A_T \rightarrow L < \infty$ as $T \rightarrow \infty$. The conditions for a *regular summability method* are given by the Silverman-Toeplitz theorem or, in terms of weighted averages, in [258]:

- $w_n \geq 0 \forall n \in \mathbb{N}$
- $m_T \rightarrow L$ as $T \rightarrow \infty$
- $w_T/W_T \rightarrow 0$ as $T \rightarrow \infty$

Case 2: $W_T \rightarrow M < \infty$ as $T \rightarrow \infty$. Here, $(w_n)_{n \in \mathbb{N}}$ is absolutely convergent, and since the metrics are bounded,

$$\sum_{i=1}^{\infty} |w_i m_i| \leq \sum_{i=1}^{\infty} U |w_i| = UM,$$

which implies

$$A_{\infty} \leq \frac{UM}{M} = U.$$

We select the weight family $w_k = k^{-\alpha}$ with $k \in \mathbb{N}^+$, resulting in the complete formulation shown in Equation 6.7. With $0 \leq \alpha < 1$ the series diverges, while if $\alpha \geq 1$ it converges absolutely. Larger α values give more importance to the earliest scores, with $\alpha \rightarrow \infty$ making $A_{\infty}^{\infty} = m_0$ (only the first score matters), while $\alpha \rightarrow 0$ approaches the model's optimum score. This behavior is illustrated in Figure 6.1.

$$A_T^{\alpha} = \frac{\sum_{r=1}^T r^{-\alpha} m_r}{\sum_{r=1}^T r^{-\alpha}} \quad (6.7)$$

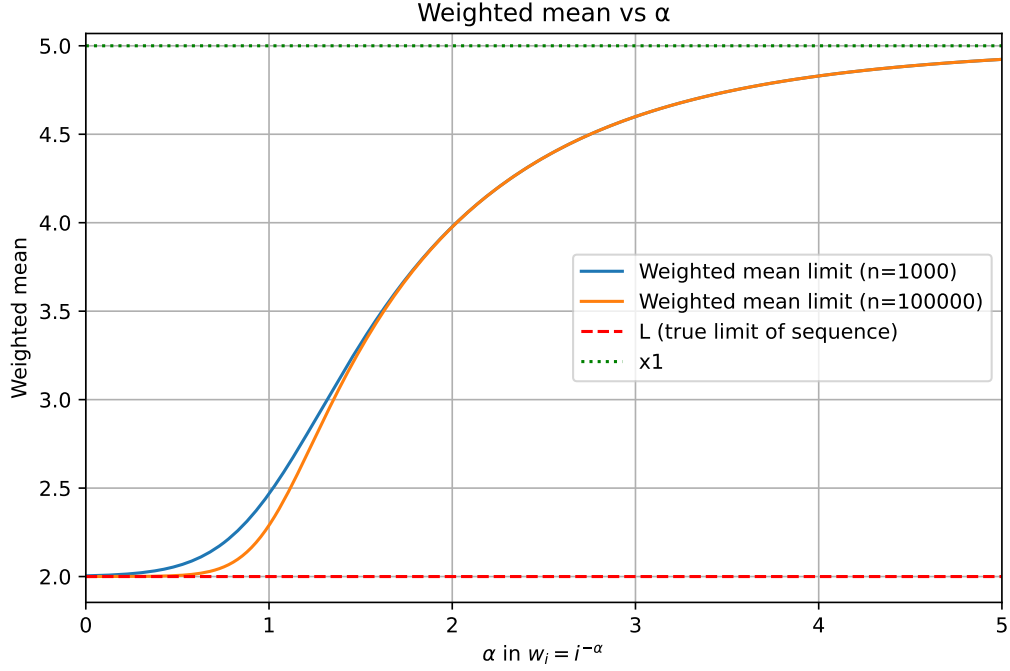


Figure 6.1: Convergence of A_∞^α for input sequence $(5, 2 + 6^{-k-1}, \dots)$ with $k \in \{1, \dots, 1000 \text{ or } 100000\}$. Truncation error causes the deviation from the true limit at $x = 1$. However, this error becomes negligible when early or convergence terms are dominant.

This formulation provides a way to explicitly control the trade-off between resources consumed and performance achieved, through the parameter α . To not overemphasize the earliest scores, we set $\alpha = \min(1, +\infty)$ in our experiments, which is the first value for which the weights series converges. This choice is coherent with the fact that collecting double the samples entails double the cost and thus its performance is roughly weighted half. Moreover, to approximate this sum with fewer experimental iterations (fewer blocks per experience), we can use the trapezoid rule to estimate the values between computed instances. This approach is particularly useful near convergence, where performance variations are minimal.

6.4.5 ABLATION STUDY

To assess the contribution of each component of our approach, we conducted an ablation study. Our model consists of three independent components:

- **force**: Forces the newly added feature to be used as the first split when growing new trees.
- **diffi**: Uses DIFFI to weight the contribution of each tree in the ensemble.
- **keepsize**: Maintains a constant forest size by replacing older trees instead of continu-

ously adding new ones.

We evaluate all combinations of these components for a total of eight configurations, including the full model and the baseline without any components. This allows us to isolate the effect of each component and understand their individual contributions to overall performance in each context.

IMPLEMENTATION DETAILS

All experiments were carried out in Python using `Scikit-learn` [259] and `NumPy` [260] for numerical operations, data standardization, and AP computation. Additionally, we used the `TreeSHAP` implementation from [261]. The proposed `RootIF` was released as open-source software¹ to foster future research, although it is not yet optimized.

We relied on the C++ implementation of `xStream` provided by the authors². The code compiles into an executable that processes concatenated training and test data in the `SVM-Lite` format and outputs anomaly scores. To ensure a fair comparison, we disabled the reset mechanism included in the implementation and ran the model in static mode. Since incremental updates are not supported, the model was retrained from scratch at each experience using all data available up to that point. The update time was measured as the difference between the total training time and a reference value recorded at the last block of every experience. This procedure can occasionally yield decreasing update times during the early blocks of some experiences due to runtime variability. Because module update and inference times are not externally measurable, we relied on the estimates reported via `stderr`.

Experiments were executed on a machine equipped with an AMD Ryzen Threadripper 1920X 12-core processor and 128 GB of DDR4 2133 MHz RAM.

6.5 RESULTS AND DISCUSSION

In this section, we present the results of our experiments. Given the large number of datasets and the complexity of the experiment, we first report selected results that highlight the key findings. The first three subsections mainly focus on the baseline comparisons and the effect of the number of newly trained trees. We report time-related metrics using the largest dataset (Mammography) to clearly observe asymptotic behavior, noting that execution patterns remain consistent across different data sources. Then we analyze how each

¹RootIF implementation is available at <https://github.com/AMCO-UniPD/RootIF/tree/publication>.

²<https://cmuxstream.github.io/>

component of our model affects AP through an ablation study. Finally, we exploit the RWP metric, which summarizes the trade-off between performance and resource consumption, to provide a comprehensive comparison of the full version of our model—training 100 new trees at each update—against the baselines and the SOTA xStream.

6.5.1 UPDATE TIME

The upper bound for this metric is given by `from_scratch`, where 100 new trees are trained at every update. The lower bound comes from `fixed`, where no new trees are trained after the first experience.

As anticipated in the complexity analysis (Section 6.3.4), the update time of our model is mainly driven by the number of new trees added at each step. Figure 6.2 shows the update time when 100 trees are trained at each update. We can see that the cost is the same in the first experience, where the model is trained like a standard iForest, and in the later updates, since the number of trees is unchanged. In comparison, Figures 6.3 shows that using only a few new trees makes updates much faster, giving us a way to adjust the cost between the two baselines depending on the application requirements.

No differences appear between the ablation variants. This is expected, since adaptive weighting only happens at inference time, and the sampling of old and new trees is negligible compared to the training cost.

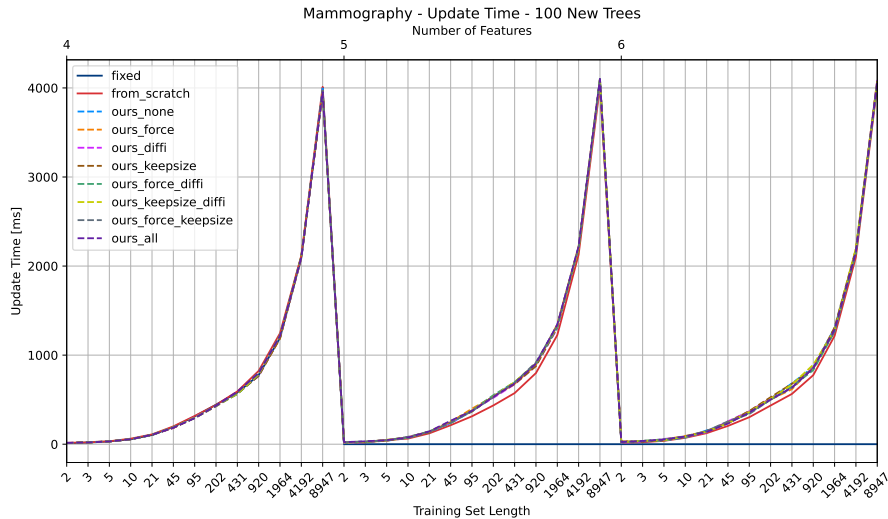


Figure 6.2: Update times for each RootIF configuration and baselines on the Mammography dataset, with 100 new trees trained per update. The x-axis displays dual metrics: the number of features (top) and the cumulative size of the training dataset (bottom) for each experience.

6.5. RESULTS AND DISCUSSION

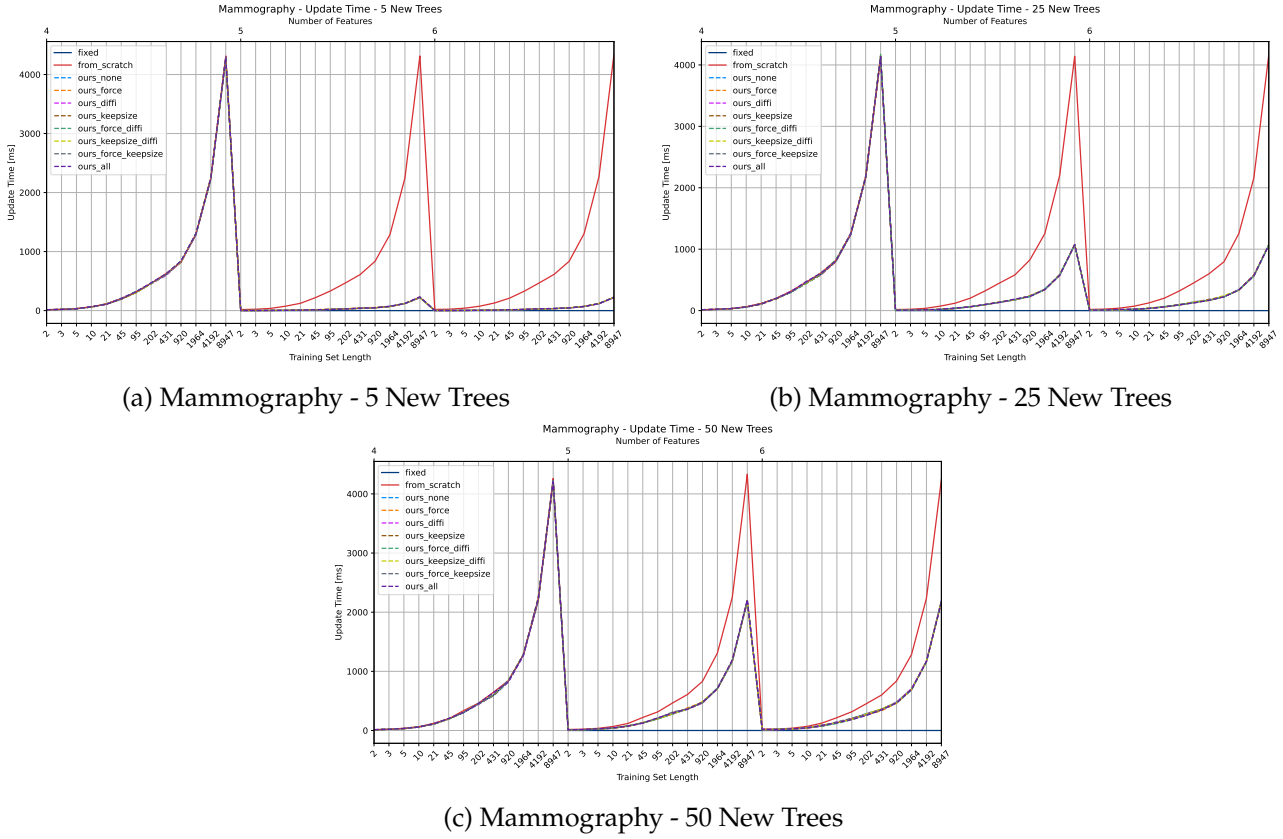


Figure 6.3: Evolution of the update time for each RootIF configuration and baselines on the Mammography dataset with different numbers of new trees trained at each update.

6.5.2 INFERENCE TIME

The interpretation of inference time is more complex. In this case, the lower bound is given by `from_scratch` model, since training on only a few samples produces shallow trees that are very fast to traverse at inference time. Conversely, `fixed` does not represent an upper bound, but rather serves as the control case of a standard iForest once the number of samples exceeds the subsampling size ϕ .

Figure 6.4 highlights an important detail: when the `keeptsize` component is used to limit the total size of the forest, inference time remains stable over time. This happens because the overall number of trees directly impacts both the traversal cost and the DIFFI computation. Among the variants without `keeptsize`, inference time increases about twice as fast when `diffi` is active compared to when it is not. This effect appears already in the first experience, since `diffi` assigns weights to every tree regardless of whether all trees see the same feature set or not.

Another reassuring result is that the `force` component has no measurable effect on

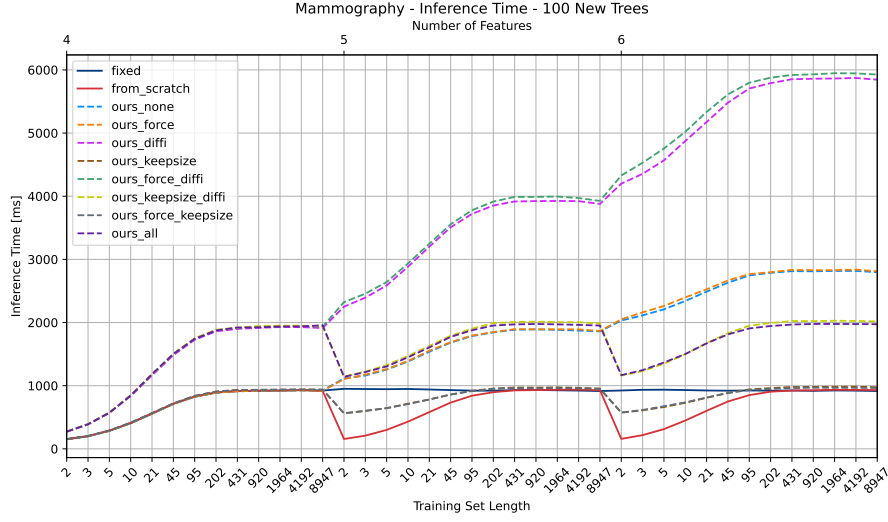


Figure 6.4: Inference time of each configuration of RootIF on the Mammography dataset with 100 new trees trained at each update.

inference time, as expected.

The behavior with `keeptsize` alone (or combined with `force`) is also consistent with our design. At the start of each experience, inference time falls roughly halfway between `from_scratch` and `fixed`, because half of the trees are fully formed and encode the old knowledge, while the newly added ones are still shallow.

Finally, Figures 6.5 show that the number of new trees mainly acts as a scale factor. Adding fewer trees at each update slows down the growth of inference time in the variants without `keeptsize`, while in the versions with `keeptsize`, fewer replacements of old trees dilute the speed-up effect of shallow new trees. The impact of `diffi`, on the other hand, is immediate and shifts inference time toward roughly 2000 milliseconds from the very first experience.

6.5.3 AVERAGE PRECISION

The quality of the anomaly scores produced by different configurations is strongly dataset-dependent, so the comprehensive evaluation is postponed to Section 6.5.4. In this section, we instead focus on the effect of the number of new trees, complementing the time-related results discussed earlier.

The hyperparameter `n_new_trees` was designed to provide a trade-off between performance and computational cost. From the previous analysis, we know that it completely determines the update time and has a strong influence on inference time, with larger values leading to higher costs. It is therefore natural to expect that adding more new trees should also improve performance, thereby balancing the increase in computational requirements.

6.5. RESULTS AND DISCUSSION

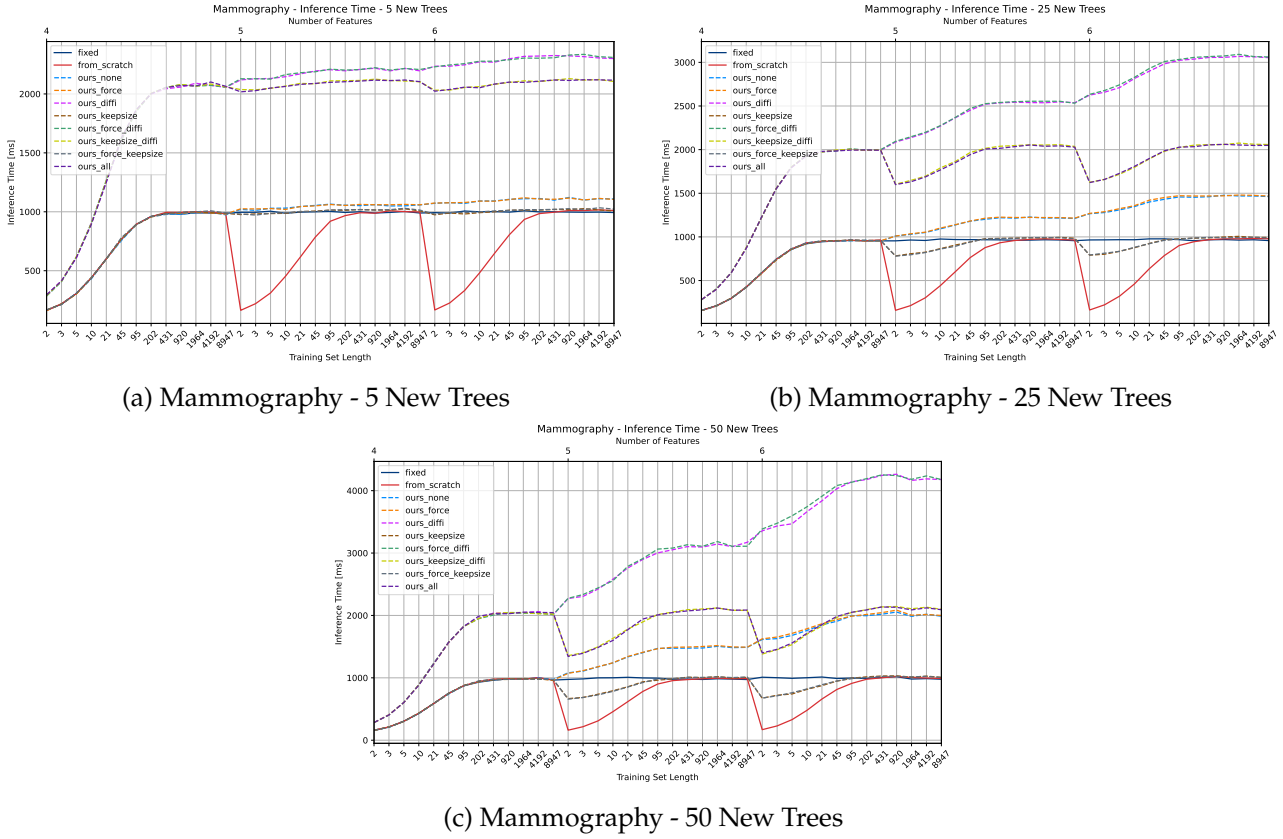


Figure 6.5: Evolution of the inference time of RootIF on the Mammography dataset with different numbers of new trees trained at each update.

A representative example of this trade-off is given by the Pima dataset. The two main objectives of our method are to adapt to the evolving feature space while retaining knowledge from previous experiences. The main mechanism that enables this balance is the combination of old and new trees. Figure 6.6 shows that, regardless of the number of new trees, the model consistently preserves old knowledge, as performance does not collapse to random levels in the first few blocks of each experience. However, the extent to which the model adapts to the new features is clearly determined by `n_new_trees`. As this parameter increases from 1 to 100, the asymptotic performance moves smoothly between the two baselines, `fixed` and `from_scratch`, with the highest values even outperforming the latter.

A noteworthy observation is that the performance of our complete model (with `keeptsize` enabled) using 100 new trees (Figure 6.6d) surpasses that obtained with 50 new trees (Figure 6.6c), even though by design both settings incorporate roughly the same number of new trees into the forest. The improvement can be attributed to the larger pool of candidate trees generated when training 100, which enhances ensemble diversity and results in more robust performance.

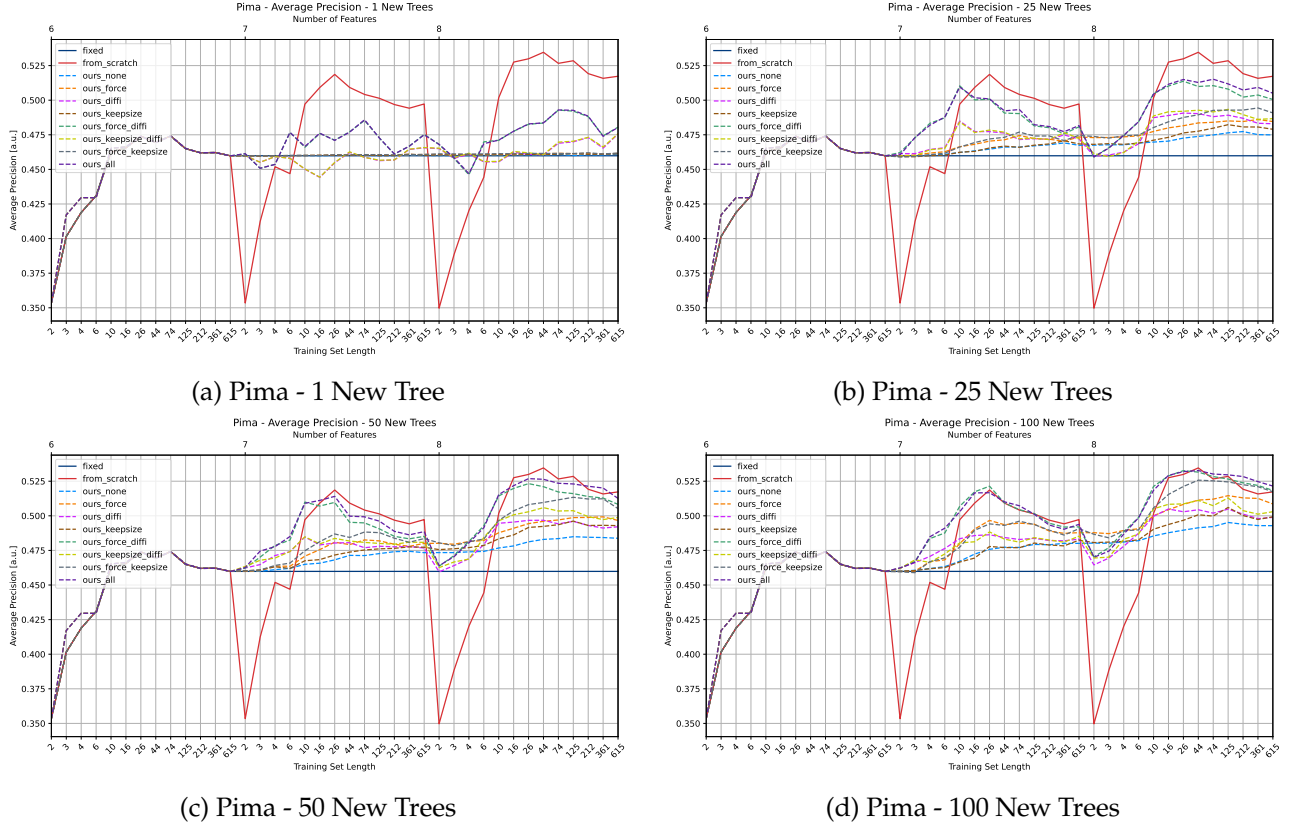


Figure 6.6: Evolution of the AP of RootIF on the Pima dataset with different numbers of new trees trained at each update.

6.5.4 COMPREHENSIVE EVALUATION

We have already discussed how the number of new trees influences both the computational cost and performance of our model. To gain a more complete picture, we now compare our method against the baselines and the SOTA, while also analyzing the contribution of each component (*keeptsize*, *force*, and *diffi*) to overall performance. In this comparison, we focus on the configuration that trains 100 new trees at each update. This choice allows us to evaluate performance on a fair basis with respect to other approaches while representing an upper bound on the computational cost, since reducing the number of new trees cannot increase either update or inference time.

Table 6.3 summarizes the results in terms of RWP, which captures the trade-off between performance and resource consumption (see Section 6.4.4). Our model, with some dataset-dependent fluctuations in the contribution of individual components, achieves the best overall performance in twelve out of seventeen datasets. This highlights its ability to balance accuracy and efficiency. Moreover, in these twelve datasets the full model configuration, with

6.5. RESULTS AND DISCUSSION

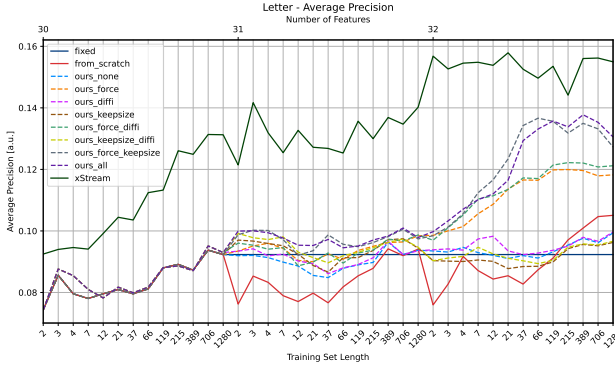
all components enabled, outperforms `xStream`.

Figure 6.7a shows the case of the Letter dataset, where `xStream` performs better than our approach. Conversely, Figure 6.7b illustrates the opposite situation on the MNIST dataset, where our model achieves substantially higher performance. In this case, most of the gain can be attributed to the force component, underscoring its effectiveness in incorporating new knowledge into the forest.

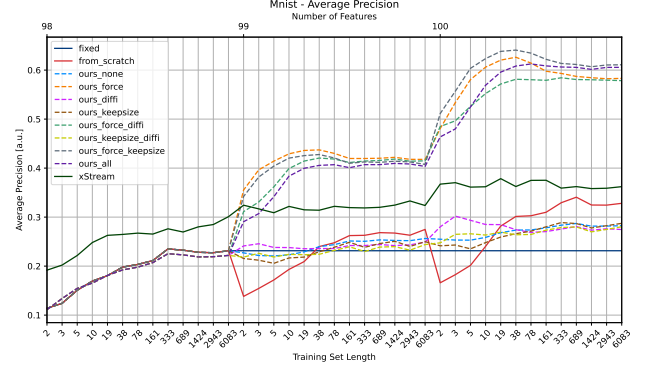
Finally, Figure 6.8 compares update and inference times on the Mammography dataset. Even in its most computationally demanding configuration (training 100 new trees per update), our model maintains a significant advantage in update time over `xStream` for nearly all training set sizes. Considering that the asymptotic complexity of RootIF is $O(T' \cdot \psi \log \psi; +; T)$ and that of `xStream` is $O(S \cdot F \cdot K \cdot m \cdot D \cdot M)$, this gap is expected to widen further as the dataset size increases. The main difference lies in the fact that RootIF adopts a subsampling strategy and remains independent of the number of features F , whereas `xStream` processes each feature for all S instances in the update. As for inference, the time required by `xStream` is almost negligible. However, in a deployment setting `xStream` would need to compute both a score and an update for each incoming instance, since it operates in an online manner.

Table 6.3: Average RWP across three experiences (best in bold), we report the baselines and our model with all the components active. We also report the performance (Abl.) and name (Abl. Name) of the best model from our ablation study.

Dataset	Abl. Name	Abl.	Ours All	Fixed	From Scratch	xStream
annthyroid	force_diffi	20.25	21.47	13.18	20.69	16.84
breastw	force	96.84	96.61	95.65	96.32	93.00
cardio	force_diffi	56.03	55.75	40.21	47.84	45.93
ionosphere	force_diffi	76.83	76.76	76.23	72.93	80.70
letter	force_keeptsize	9.61	9.66	8.62	8.74	12.53
mammography	force_diffi	14.18	16.07	5.29	13.24	17.81
mnist	force_keeptsize	41.92	41.12	23.26	27.28	32.53
optdigits	force_keeptsize	7.52	7.63	3.78	4.39	2.80
pendigits	force_diffi	26.50	26.29	20.23	24.58	20.13
pima	force_diffi	51.27	51.31	50.22	51.25	47.67
satellite	diffi	66.54	66.17	66.33	66.23	63.24
satimage-2	force_diffi	92.33	92.26	91.39	90.66	92.14
thyroid	force_diffi	35.89	37.52	15.96	34.64	28.06
vertebral	force_keeptsize	11.74	11.75	12.20	11.60	12.02
vowels	force_diffi	14.39	14.15	8.12	10.58	18.30
wbc	force	69.18	68.43	64.54	64.80	62.50
wine	force_keeptsize	34.38	36.51	17.56	29.37	25.15

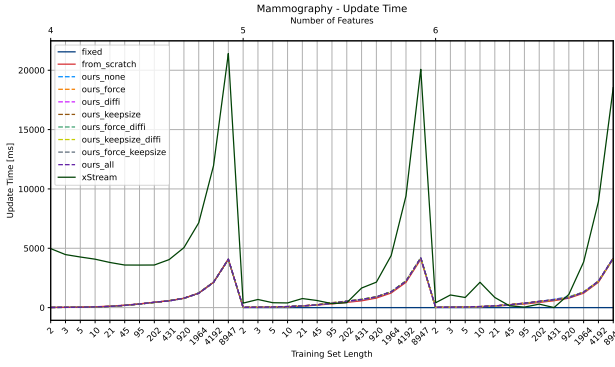


(a) Average Precision on the Letter dataset.

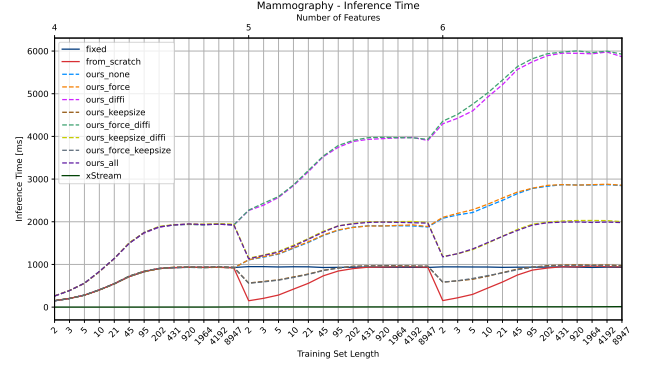


(b) Average Precision on the MNIST dataset.

Figure 6.7: Examples of our model outperforming and being outperformed by xStream.



(a) Update Time on the Mammography dataset.



(b) Inference Time on the Mammography dataset.

Figure 6.8: Comprehensive comparison of update and inference time on the Mammography dataset.

CONSTANT FOREST SIZE

The effect of the keeptime component is primarily related to long-term computational costs, as it stabilizes inference time by maintaining a constant number of trees in the forest. Beyond this efficiency aspect, it also brings a modest performance improvement on several datasets, as shown in Table 6.4. This gain can be attributed to the regular replacement of older trees with newly trained ones, which enables the model to adapt to an evolving feature space more easily.

FORCING ROOT SPLITS

The force component consistently improves performance across most datasets, with gains of up to 16% points in RWP, as reported in Table 6.4. This benefit likely stems from the more effective integration of new features: by forcing splits on these features at the root level, every tree is guaranteed to account for them. This mechanism allows the model to

6.6. SUMMARY OF FINDINGS

quickly adapt to changes in the feature space. Further evidence is provided in Table 6.5, which shows that the improvement is larger when feature importance is concentrated on the newly introduced features (i.e., the last ones in reverse TreeSHAP order). In contrast, when importance is spread more evenly across all features, the contribution of this component is marginal.

DIFFI ADAPTIVE WEIGHTING

At first glance, the results in Table 6.4 suggest that the `diffi` component has a modest impact on overall performance, with clear gains emerging only on a few datasets. However, this perspective overlooks the dynamic role of the adaptive weighting mechanism. Its main advantage becomes evident in situations where the newly added trees are too few or too weak to dominate the ensemble, which typically occurs with small dataset sizes or when only a few trees are added per update. This behavior can be observed, for example, in Figure 6.7b, where `diffi` produces a noticeable peak in performance around the three-sample mark on the lower tracks, or in Figure 6.6b, where it provides a similar localized boost.

6.6 SUMMARY OF FINDINGS

In this chapter, we investigated the challenge of anomaly detection in feature-evolving environments, where the dimensionality of the data grows over time as new sensors or data sources become available. Traditional models, such as the widely used iForest, are not designed to handle this dynamic setting: retraining from scratch discards prior knowledge and incurs prohibitive computational costs, while choosing to not update the model prevents it from exploiting new information leaving the system vulnerable to unnoticed faults. Our objective was to design a method capable of incorporating new features while retaining knowledge encoded in past models with minimal computational overhead or data requirements.

Our main contribution was the development of *Root-forced Isolation Forest*, an extension of the classical iForest that explicitly addresses feature-evolving scenarios. The method combines three key procedures: (i) construction of new trees based on the new dataset, where the initial splits are forced on the newly introduced features; (ii) merging of new and old trees from the original forest, with the possibility of pruning to control the final forest size; and (iii) adaptive score aggregation based on DIFFI local feature importance, which balances the contributions of historical and new trees depending on the relevance of their split features to each sample. Together, these components allow the model to adapt to an evolving feature

Dataset	force	keepsizes	diffi
annthyroid	3.03	1.14	1.49
breastw	0.10	-0.07	0.07
cardio	11.19	0.64	0.37
ionosphere	1.06	-0.05	0.12
letter	1.15	0.32	0.07
mammography	5.38	1.48	2.70
mnist	16.25	0.15	-0.72
optdigits	3.30	0.37	0.13
pendigits	5.87	0.10	0.33
pima	0.52	-0.14	0.08
satellite	-0.44	-0.04	0.07
satimage-2	0.57	-0.24	0.12
thyroid	8.72	2.48	3.51
vertebral	-0.00	0.17	0.01
vowels	5.11	0.33	0.52
wbc	2.65	0.05	0.04
wine	13.86	1.77	2.10
Average	4.61	0.50	0.65

Table 6.4: Average improvement in terms of RWP across the three experiences and four component combinations. The values represent the difference in RWP when the component is enabled compared to when it is disabled (e.g. "force_diffi - force" is one of the four terms to average for diffi), with positive values indicating an improvement.

Dataset	AP difference
annthyroid	0.24
breastw	-0.00
cardio	0.20
ionosphere	0.02
letter	0.01
mammography	0.24
mnist	0.10
optdigits	0.01
pendigits	0.11
pima	0.03
satellite	0.01
satimage-2	0.01
thyroid	0.46
vertebral	0.00
vowels	0.09
wbc	0.04
wine	0.39
-	-

Table 6.5: Average AP difference between removing the first and last two features in TreeSHAP order of importance. Positive values indicate that certain features are more discriminative for anomaly detection.

space while preserving prior knowledge and keeping computational requirements limited.

We validated the proposed approach through extensive experiments on seventeen real-world datasets from the ODDS repository, covering a wide range of domains and dimensionalities. The results demonstrated that our model consistently outperforms standard iForest baselines and even outperforms the SOTA online method xStream on fourteen datasets. The ablation study highlighted the individual contribution of each component: `force` ensured effective integration of new features, `keepsizes` limiting asymptotic update and inference computational complexity while slightly improving performance, and `diffi` provided localized gains in low-data or low-tree regimes.

It is important to note certain limitations of this work. Our implementation of the proposed method was a research prototype in Python and is not yet optimized for production use. Despite this, the results show that it is already comparable with xStream while far from an optimized iForest implementation. Moreover, our approach relies on batch-incremental

6.6. SUMMARY OF FINDINGS

updates, where the model is retrained and redeployed at discrete intervals, rather than operating in a fully online manner. In the next section we will discuss potential future research directions to address these limitations and further enhance the method.

6.6.1 DIRECTIONS FOR FUTURE RESEARCH

Building on the contributions of this thesis, several promising avenues remain for future work.

A primary direction for future work is extending RootIF to a fully online setting. This entails developing strategies to dynamically identify and replace redundant trees. Such a mechanism would naturally facilitate the pruning of uninformative features, thereby relaxing the current assumption that all newly added sensors are informative.

A second path is to explore different local feature importance methods. While DIFFI provides a solid and efficient foundation, its main problem is that it is not additive like SHAP values, which would allow a more robust aggregation of scores.

It would also be beneficial to investigate prioritization methods that extend beyond the root node. By allowing for more feature diversity in the paths, these methods could further increase the performance improvements provided by the adaptive score aggregation.

Finally, scaling the method to industrial deployment requires optimized implementations, potentially leveraging parallel training of isolation trees or GPU-based acceleration. Such improvements would bring the computational performance closer to production-ready standards without sacrificing the methodological advantages demonstrated in this thesis.



Robust Fault Detection And Identification

The critical importance of Fault Detection and Identification (FDI) in industrial scenarios has been thoroughly established throughout this thesis. While Fault Detection is essential for recognizing hazardous situations, Fault Identification is what transforms a general alert into an actionable diagnosis. By pinpointing the exact nature of a failure, FDI allows for targeted interventions. This significantly reduces downtime and optimizes resource allocation, ensuring both system safety and operational reliability.

Most FDI approaches found in the literature, some of which were mentioned in this thesis, were developed in general-purpose contexts and show strong performance when trained on datasets with well-defined properties, such as independent and identically distributed samples, correct labeling, and balanced classes. In practice, however, industrial scenarios face the challenges described in Section 2.5, which can greatly reduce the effectiveness of traditional models.

In this chapter, we focus on two of these challenges: *nominal data contamination* and *fault data scarcity*. To overcome these difficulties, we propose the innovative Forest of Local EXpert Classifiers (FLEX-C) method, a family of semi-supervised structured ensemble approaches which proves robust to these conditions.

In real industrial FDI, systems are core parts of the value chain and need continuous monitoring to ensure reliability. The data collected mostly represent nominal operating conditions, while faults are rare. As a result, datasets are strongly imbalanced, with only a small portion corresponding to faulty situations. In addition, the data consist of raw sensor readings, and labeling requires expert input. Since labeling is costly, only serious faults are usually recorded, while minor or transient anomalies and nominal data remain unlabeled. This means that datasets are not only imbalanced but also that the *no-fault* class can include

7.1. PROXY TASK

misabeled samples, which we call *nominal data contamination*. Both of these issues are known to degrade the performance of traditional supervised models [262, 263, 264, 265].

While the literature offers solutions for each problem individually, the combination of the two has received little attention. This is the gap we aim to address.

Our contributions are summarized as follows:

- We analyze the FDI problem under the combined challenges of *nominal data contamination* and *fault data scarcity*.
- We introduce a family of semi-supervised structured ensemble models, FLEX-C, designed for greater robustness under these conditions.
- We implement and evaluate two variants of this family: FLEX-C Labels and FLEX-C Centroids.
- We perform a systematic comparison against RF, widely regarded as SOTA for the datasets under study, as well as other baselines.
- We demonstrate that, within the range of supervised training dataset sizes realistically available in industrial scenarios, FLEX-C matches, and in some cases surpasses, the performance of RF trained on cleaned and balanced datasets.

The remainder of this chapter is structured as follows: Section 7.1 explains how this work, which uses publicly available datasets for broader reproducibility, relates to FDI in the hydroelectric context. Section 7.2 reviews the relevant literature. Section 7.3 introduces our approach, the models we use, and the related work it builds on. Section 7.4 describes the datasets, preprocessing, and evaluation procedure. Section 7.5 presents the experimental results. Finally, Section 7.6 concludes with the main findings and future research directions.

7.1 PROXY TASK

With this work, we extend the idea behind Active Learning Isolation Forest (ALIF) to the classification task by introducing FLEX-C, a family of semi-supervised structured ensemble learning methods. These methods leverage the flexibility and robustness of the unsupervised AD model iForest to partition the input space, thereby simplifying the classification problem. The main idea is to use iTrees to divide the input space into regions, and then train a base classifier in each region. The final prediction is obtained by aggregating the outputs of the base classifiers.

We tested two variants of FLEX-C, namely FLEX-C Labels and FLEX-C Centroids, on unbalanced classification datasets as proxy tasks for fault detection and identification in HPPs. Through our experimental setup these datasets reflect the industrial scenario where data is strongly unbalanced towards normal samples, and only a small number of labeled fault instances are available. Since labeling is costly, normal samples are often not verified and may contain undetected faults, which reduces the performance of supervised models. Our results show that, within the typical range of labeled dataset sizes available in practice, FLEX-C outperforms Random Forest (RF) trained on a balanced and cleaned version of the dataset.

Furthermore, as an extension of ALIF, FLEX-C can naturally be combined with Active Learning (AL) in future developments. This would allow users to interactively redefine what constitutes a fault, add new fault types, and correct misclassified samples. Such capabilities represent a further step toward the objectives defined in Section 5.1.

7.2 BACKGROUND AND RELATED WORK

In the literature, there is no universally established term for the problem setting considered in this chapter. Nevertheless, under the broader categories of *Imbalanced Data* and *Noisy Labels*, several approaches have been proposed that are relevant to our work. This section provides an overview of these approaches.

7.2.1 IMBALANCED DATA AND FAULT DATA SCARCITY

If the dataset is assumed to be fully labeled, meaning that the data representing the nominal working condition of the system does not contain unnoticed faults, then the challenge of *Fault Data Scarcity* can be regarded as a specific instance of the broader problem of *Imbalanced Data*, where one or more classes (fault types) have significantly fewer examples than others (nominal operation). The literature offers a variety of techniques to address this issue.

Resampling methods. The most direct way to address class imbalance is to modify the training data distribution through resampling techniques. Oversampling methods such as the Synthetic Minority Oversampling Technique [266] and Adaptive Synthetic Sampling [267] generate synthetic samples of the minority class to balance the dataset. Undersampling methods, including Tomek Links [268] and Edited Nearest Neighbors [269], instead reduce the number of majority class samples, retaining only the most informative ones. Hybrid

approaches combine both oversampling and undersampling to achieve a more balanced and representative training set.

Anomaly Detection approaches. In binary classification settings (fault vs. normal), one can also leverage approaches from AD. These methods, detailed in Section 2.4.6, aim to identify deviations from the nominal condition and have been extensively discussed throughout this thesis.

Semi-Supervised Learning. Semi-supervised learning techniques are effective strategies for addressing class imbalance, particularly in scenarios where labeled fault data is scarce. Our proposed method belongs to the semi-supervised category. They leverage the unlabeled data by combining it with the limited labeled set during training, often through:

- **Consistency regularization** approaches use regularization terms that encourage the model to produce consistent predictions for unlabeled data under small perturbations or augmentations. A relevant example is [270], where the authors use an autoencoder to learn a latent representation of the data and apply consistency regularization to improve fault diagnosis performance with limited labeled data.
- **Pseudo-labeling** generally consists in training a model on the labeled data, using it to predict pseudo-labels for the unlabeled data, and then retraining the model using also the more confident pseudo-labels. This can be done iteratively to refine the pseudo-labels. A relevant example of this approach is [271], in which the authors iteratively train a RF model on unlabeled data starting from a small labeled dataset.
- **Label propagation** methods construct a graph where nodes represent data points and edges represent similarities between them. Labels from the labeled nodes are then propagated to the unlabeled nodes based on their connections. An example of this approach is [272].

Few-Shot Learning. This paradigm relies on transfer learning and meta learning techniques to allow models to learn new classes from only a few labeled examples. The core idea is to exploit prior knowledge from related tasks or domains to generalize effectively to unseen classes with minimal supervision. However, this approach is not well suited to our current scenario, as it requires a substantial amount of labeled data from related domains, which is not available. Nevertheless, as more fault diagnosis data becomes accessible in the future, this paradigm could become a strong candidate for further exploration. Relevant examples can be found in [273, 274].

7.2.2 NOISY LABELS

There is no universally adopted strategy for addressing the problem of *Noisy Labels*, but several approaches have been proposed in the literature, some of which are also robust to class imbalance due to their semi-supervised nature. In [275], the authors employ a Gaussian Mixture Model to separate clean from noisy samples and then train a pair of deep neural networks using the MixMatch semi-supervised learning framework [276], which combines consistency regularization with a pseudo-labeling procedure. In [277], a Generative Adversarial Network (GAN) is used to generate synthetic samples, and a classifier is trained jointly on both real and synthetic data.

Active Learning (AL) techniques can also mitigate the impact of *Noisy Labels*. They typically consist of two components: a query strategy, aimed at selecting the most informative samples and a learning algorithm, which updates the model with the newly acquired labels [154]. Since labeling an entire dataset often involves redundant effort on uninformative data, AL allows to drastically reduce the burden on domain experts by isolating only the most informative samples for review. In [278], the authors propose an approach in which potentially mislabeled samples are removed based on the influence of the model on the sample, while query points are selected according to the influence of the sample on the model. In this context, influence is measured as the disagreement between the model's predictions.

7.3 METHODOLOGY

This section presents the methodology of our research on FDI in an industrial scenario, focusing on the challenges of *Nominal Data Contamination* and *Fault Data Scarcity*. To address these issues, we propose a new family of methods, Forest of Local EXpert Classifiers (FLEX-C). Section 7.3.1 introduces the RF model, used as our main comparison with the SOTA. Section 7.3.2 outlines the general FLEX-C approach and the concepts that inspired it, while Section 7.3.3 details our two implementations, FLEX-C Labels and FLEX-C Centroids, including their underlying principles, algorithms, and computational complexity. Since the description of our approach is extensive and the experiments are closely tied to practical considerations, the experimental setup is presented separately in Section 7.4.

7.3.1 RANDOM FOREST

RF [81] is a versatile and powerful ensemble learning method applicable to both classification and regression tasks. It constructs an ensemble of Classification and Regression

7.3. METHODOLOGY

Trees (CART) [279] during training and outputs either the majority class among the trees for classification or the average prediction for regression.

ALGORITHM DESCRIPTION

The algorithm's strength lies in the introduction of randomness in two key stages of the construction of the forest, which helps to reduce the variance of the model without a substantial increase in bias.

Training A RF model is composed of a collection of t individual decision trees. Each tree in the forest is trained on a different random subset of the data and features, following these steps:

1. **Bootstrap Aggregating (Bagging):** Each of the t trees is trained with a random bootstrap sample drawn from the original training dataset. This sample has the same size as the original dataset but is selected with replacement, meaning some instances may appear multiple times while others may be omitted entirely. The instances that are not included in the bootstrap sample are referred to as "Out-Of-Bag" (OOB) samples and can be used for internal model validation.
2. **Feature Randomness:** When building each decision tree, at every node, instead of searching for the best split among all available features, the algorithm selects a random subset of features and searches for the optimal split only within that subset. The size of this feature subset, denoted as K , is a key hyperparameter. A typical value for classification is $\sqrt{F}$, where F is the total number of features.

This combination of bagging and feature randomness ensures that the individual trees are diverse, and their errors are largely decorrelated, which is critical to the performance of the ensemble.

Prediction Once the forest is trained, making a prediction for a new instance x is straightforward. The input instance is propagated through every tree in the forest, with each tree independently generating a classification. The final prediction is determined by a majority vote among all the trees. The class that receives the most votes is chosen as the final label for the instance.

Accordingly, the final prediction $\hat{y}$ for a classification task can be formally expressed as:

$$\hat{y} = \operatorname{argmax}_{c \in C} \sum_{t=1}^T I(\hat{y}_t(x) = c) \quad (7.1)$$

where T is the total number of trees, $\hat{y}_t(x)$ is the prediction of the t -th tree for instance x , C is the set of all possible classes, and $I(\cdot)$ is the indicator function, which is 1 if the condition is true and 0 otherwise.

TIME COMPLEXITY

The computational complexity of RF is a critical factor in its practical application, especially when working with large datasets.

Training Time Complexity The training process involves building T decision trees. Let S be the number of training instances and F be the total number of features. The computational complexity to build a balanced decision tree is $O(N \log N \cdot K)$, where K is the size of the random feature subset considered at each node. Since the algorithm builds T such trees, the total training time complexity is given by:

$$O(T \cdot K \cdot S \log S) \quad (7.2)$$

As T and K are hyperparameters that do not scale with N or F , the training complexity is often considered to be effectively linear with respect to the number of training instances.

Inference Time Complexity During inference, each of the test instances must be passed through all T trees in the forest. The time to make a prediction for a single instance in one tree is proportional to the depth of the tree, which for a balanced tree is on the order of $O(\log S)$. Therefore, the per sample inference time complexity:

$$O(T \cdot \log S) \quad (7.3)$$

This makes the prediction process highly efficient and scalable, as its complexity is linear in the number of test instances.

7.3.2 FOUNDATIONAL METHODS AND ADAPTATION

The proposed FLEX-C framework draws inspiration from the active learning strategies in ALIF [168] and BALIF [169]. We translate these concepts (originally designed for AD) to the domain of FDI.

These methods leverage the tree-based structure of iForest to identify uncertain or informative instances for labeling, allowing efficient querying in data-scarce scenarios. A key advantage is that user input can efficiently alter the depth of the leaves in the trees, thereby

7.3. METHODOLOGY

correcting the anomaly scores and, by extension, the overall predictions of the iForest. This structural feedback mechanism enables rapid model adjustment with minimal supervision, making the approach highly suitable for interactive learning under uncertainty.

In FLEX-C, we extend this concept to the industrial FDI setting. Instead of relying exclusively on unsupervised anomaly scores, FLEX-C integrates an ensemble of weak supervised classifiers within a hybrid decision framework that combines structural properties of iForest with supervised feedback. This design allows the method to learn effectively with the limited available fault samples, while leveraging the inherent robustness of iForest to effectively handle the contamination in the normal class efficiently.

ALIF

In the following, we highlight the core concepts of ALIF, preferred over BALIF since it allows for a simpler and more direct explanation, and serves as the basis for our proposed methods.

As reported in its original paper [168], ALIF is an AL-based approach for detecting anomalous behavior which requires a restricted number of labels and adapts the detector to align with the user's definition of anomaly. Enabling AL typically requires two key components: a *query strategy*, which selects a small and informative subset of data points to be labeled by the user in order to improve model performance; and a *retraining strategy*, which incorporates user feedback to efficiently modify the model's predictions.

Although the former could be relevant in future extensions of this work, as discussed in Chapter 7.6.1, it is the latter that constitutes the core idea behind both ALIF and our proposed method. ALIF introduces a simple retraining strategy (later refined in BALIF) [169] that allows for adjusting the depth of the leaves in the iTrees, thereby directly influencing the computation of anomaly scores based on user feedback.

More explicitly, let L denote a leaf node in any binary decision tree within the iForest. ALIF assigns a *leaf coefficient* value $k(L)$ to each leaf, defined as:

$$k(L) = \frac{1}{2} \left(\frac{L_o - L_i}{L_o + L_i} + 1 \right)$$

where L_o and L_i represent the number of supervised outliers and inliers, respectively, that reach leaf L in the tree. Intuitively, if the majority of supervised instances that fall into a leaf are outliers, then $k(L)$ approaches 1; if they are mostly inliers, $k(L)$ approaches 0.

When at least one supervised instance reaches a leaf, ALIF overwrites the standard iForest depth with a supervised-adjusted depth. Two alternative strategies are proposed:

1. Piecewise Linear Supervised Depth:

$$h^s(k) = \begin{cases} 2k [c(\psi) - h_{\max}] + h_{\max}, & \text{if } 0 \leq k < \frac{1}{2} \\ 2k [h_{\min} - c(\psi)] + 2c(\psi) - h_{\min}, & \text{if } \frac{1}{2} \leq k \leq 1 \end{cases} \quad (7.4)$$

2. Logarithmic Supervised Depth:

$$h^s(k) = -c(\psi) \log_2(k) \quad (7.5)$$

where $c(\psi)$ is the average path length for unsupervised data of size ψ , and $h_{\min}$, $h_{\max}$ are the minimum and maximum depths in the trained iForest.

The purpose of these functions is to reduce the depth of leaves associated with many supervised outliers, thereby increasing their anomaly scores, while increasing the depth for leaves with many inliers, thus reducing their likelihood of being flagged as anomalies. We do not explore these functions in detail, as they are not central to this work. However, the original paper provides further considerations regarding their bounds and stability.

GENERALIZATION AND ADAPTATION

ALIF's *retraining strategy* can be interpreted at a higher level as assigning an extremely weak regressor to each leaf. Specifically, the regressor $h^s(k)$, with a single degree of freedom given by the *leaf coefficient* $w = k(L) \in \mathcal{R}$, is used to approximate the depth of an idealized tree—one that shares the same structure but belongs to an iForest perfectly aligned with the user-defined notion of normality.

The structure of each tree of the iForest then acts as a partitioning of the input space, enabling its leaves to serve as a local expert. The final anomaly score aggregates the predictions of these weak regressors across the forest, effectively capturing localized feedback and propagating it through the ensemble.

In this work, we adapt these concepts to the classification setting by associating a weak classifier with each leaf, rather than a regressor. Since we are addressing a classification problem, we cannot rely on the anomaly score computation to aggregate predictions as in the original ALIF framework. Instead, we propose tailored aggregation functions for each of our models, designed to combine the outputs of the weak classifiers in a way consistent with the principles previously defined.

LIMITATIONS

The original framework operates in a highly advantageous scenario:

- It has a default initialization for the regressors (the original leaf depths) and can correctly perform inference immediately after fitting the iForest. Conversely, there is no straightforward way to initialize a classifier meaningfully without knowing which labels correspond to faults and without having some examples indicating the location of such anomalies within each subspace.
- The standard anomaly score already serves as a valid aggregation function.
- The complexity at the leaf level remains constant in AD, as each leaf is only required to output a single scalar value. In contrast, a weak classifier must generate a probability distribution over N classes, meaning the output dimension (and thus the complexity) scales linearly with the number of potential faults.

Overall, both methods are subject to strong regularization imposed by the coarse structure of the iForest, determined by its maximum depth set via the *max_samples* parameter. On one hand, this keeps the complexity in check, reducing overfitting and the amount of supervised data required. On the other hand, it limits the expressivity of the composite model. With deeper trees, leaves represent smaller subspaces, allowing for more specialized weak models.

In this study, we did not explore this aspect further. As explained in 7.4, data that is representative for this scenario is difficult to find and generally contain few samples. Therefore, we preferred to focus on verifying that the performance of our models aligns with our expectations while effectively addressing the specific challenges of our scenario.

7.3.3 FLEX-C METHODS

We have already introduced the generalization that forms the core idea of our work. Building upon this foundation, we now describe the specific method proposed in this thesis, which constitutes a concrete implementation of that idea tailored to the requirements of our scenario.

- **Normal Data Contamination:** To address this challenge, we exploited the properties of the underlying iForest, assigning weak classifiers only to the shallower leaves. In industrial settings, the ratio of normal to anomalous data (called contamination in most libraries) can often be estimated and used to adjust the threshold of the iForest *decision function*. In our experimental design (Section 7.4), where the contamination level is set

to 10%, classifiers were assigned only to leaves whose individual tree anomaly scores exceeded the estimated threshold. Moreover, we introduced a user-defined margin parameter that fine-tunes the allowable leaf depth for classifier assignment, providing further control over the contamination-handling mechanism.

- **Labeled Data Scarcity:** To reduce complexity, we chose to use very simple classifiers, allowing the system to be deployed with only a small amount of user input. This design choice also helps prevent overfitting and leaves the door open for future integration of active or incremental learning capabilities, as will be discussed in Section 7.6.

Finally, considering that the unsupervised base structure is an iForest and that the weak classifiers are local experts within the subspace defined by their corresponding leaves, we refer to our family of methods as FLEX-C.

FLEX-C LABELS

FLEX-C Labels represents the most straightforward implementation of our core concept. Essentially, each classifier assigns a single label to all points that fall into the corresponding leaf. To determine the label from the supervised training samples, we used Algorithm 3.

Note that this update procedure is not well suited for small successive incremental updates, as the mode of separate datasets may not be the mode of the combined dataset. However, in our experiments, we did not include incremental learning challenges, so this approach remains sound. That said, it would not be difficult to improve this method using well-known algorithms for identifying the most frequent element in a data stream.

Algorithm 3 Update Method

Require:

Leaf: The leaf classifier to update
X: A batch of feature vectors.
Y: Corresponding labels for the vectors in *X*.

- 1: **procedure** UPDATE(*Leaf*, *X*, *Y*)
 - 2: *Leaf.label* $\leftarrow$ MODE(*y*) ▷ Assign the most frequent label
 - 3: **end procedure**
-

To merge the predictions of each weak classifier we again used a *mode* operation, as shown in Algorithm 4. Additionally, we return the relative count as a measure of confidence. Note that leaves without a label return *NaN* (Not a Number), which is propagated through the *mode* operation. This behavior reflects the fact that if the weak classifier never received an

Algorithm 4 Predict Labels

Require:

X: A batch of feature vectors.

```

1: procedure PREDICTLABELS( $X$ )
2:   Initialize  $L \leftarrow$  empty matrix
3:   for each tree  $t$  in Isolation Forest do
4:      $Leaves \leftarrow t.PREDICT\_LEAVES(X)$  ▷ Traverse the tree to find the leaves
5:     Initialize  $Labels \leftarrow$  empty list
6:     for each leaf  $f$  in  $Leaves$  do
7:        $Labels.APPEND(f.label)$ 
8:     end for
9:     Append  $Labels$  as a column to  $L$ 
10:  end for
11:   $(mode, count) \leftarrow MODE(L, \text{along rows})$ 
12:  return  $(mode, count/\text{number of trees})$ 
13: end procedure

```

update, then it is likely associated with a leaf containing normal samples. After *Predict_labels*, it is straightforward to replace the *NaN* values with the normal label 0.

Finally, we analyze the computational complexity of our method. This is particularly important because our experiments compare our research prototype against optimized, production-grade libraries for FLEX-C and iForest. A theoretical complexity analysis provides a fair baseline for evaluating performance, independently of implementation-level optimizations.

Training Complexity The training process can be divided into two main phases. We will use T for the number of trees, ψ for the subsampling size per tree, and S for the number of labeled samples used for supervision.

1. **Building the Forest Structure:** First, we build the iForest. As discussed in Section 4.1.1, this has a complexity of $O(T\psi \log \psi)$. Next, we attach a data structure to each leaf to hold a classifier. This requires visiting all nodes in each tree to find the leaves. Since a binary tree of depth $\log_2 \psi$ has at most $O(\psi)$ nodes, this step takes $O(T\psi)$ time.
2. **Injecting Supervised Knowledge:** Once the forest is built, we incorporate the S supervised samples. The first step for each sample is to find its corresponding leaf in every tree, which takes $O(ST \log \psi)$. After the samples are assigned to leaves, we update the classifiers. The update relies on computing the mode, which is a linear operation. Aggregated across all leaves, the total time required to update the classifiers with S samples is $O(S)$, regardless of how the samples are distributed among the leaves.

Combining these costs, the overall training complexity is dominated by the construction of the forest and the traversal of the forest with the supervised data, resulting in $O(T\psi \log \psi + ST \log \psi)$. The second term represents the computational cost required to integrate the knowledge from S data points into the tree-based structure.

Inference Complexity To make a prediction for a new, single sample, the process is as follows:

1. The sample is passed through each of the T trees to identify its leaf. Since each tree has a depth of $O(\log \psi)$, this step takes $O(T \log \psi)$.
2. The T labels from the resulting leaves are collected, and the final prediction is determined by finding the mode of these labels, which costs an additional $O(T)$.

The total time for inference is the sum of these steps, $O(T \log \psi + T)$. Because the $O(T \log \psi)$ term dominates the $O(T)$ term, the complexity of the overall inference simplifies to $O(T \log \psi)$. This corresponds to the same per sample asymptotic complexity as that of a standard iForest.

FLEX-C CENTROIDS

With FLEX-C Centroids, we increase the complexity of the weak classifiers: each leaf is now associated with a list of centroids, one for each class that has at least one sample reaching that leaf during the injection of supervised information from the user. This naturally increases the computational cost, but it also enables a more fine-grained representation of each class within a leaf.

The update procedure either adds a new centroid when it encounters an unseen class, or adjusts the centroid of an already seen class by incorporating all points observed since the classifier's initialization. The full procedure is shown in Algorithm 5. We maintain the centroid as the average of all points seen for that class. Using an iterative formulation of the average, we do not need to store all previous samples in memory, only the current centroid and the count of points it represents are required.

The prediction procedure must pass to the aggregation function the distances between the input sample and each centroid associated with the labels stored in the classifier. In Algorithm 6, we provide a simplified version of the procedure for a single sample. While the version used in our implementation is less readable, it is significantly faster. To achieve this

efficiency, we leverage the Euclidean distance identity:

$$D_j = \sqrt{\|x\|^2 + \|c_j\|^2 - 2x \cdot c_j}$$

Although this formulation does not reduce the asymptotic complexity, it avoids expensive nested loops by replacing them with highly optimized matrix operations. This results in considerable practical speed improvements, especially when processing large batches of samples.

Algorithm 5 Incremental Centroid Update

Require:

Leaf: The leaf classifier to update
X: A batch of feature vectors.
Y: Corresponding labels for the vectors in *X*.

```

1: procedure UPDATE(Leaf, X, Y)
2:   labels  $\leftarrow$  set(Y)                                      $\triangleright$  The labels in Y
3:   count  $\leftarrow$  COUNT(X, Y, labels)                      $\triangleright$  The number of points for each label
4:   sum  $\leftarrow$  SUM(X, Y, labels)                           $\triangleright$  The sum of the points with same label
5:   for i  $\leftarrow$  0 to |labels| - 1 do
6:     label  $\leftarrow$  labels[i]
7:     if thennot Leaf[label]
8:       Leaf[label].centroid  $\leftarrow$   $\frac{\textit{sum}}{\textit{count}}$ 
9:       Leaf[label].n  $\leftarrow$  count
10:    else
11:      Leaf[label].centroid  $\leftarrow$   $\frac{\textit{Leaf}[\textit{label}].\textit{centroid} * \textit{Leaf}[\textit{label}].\textit{n} + \textit{sum}}{\textit{Leaf}[\textit{label}].\textit{n} + \textit{count}}$ 
12:      Leaf[label].n  $\leftarrow$  Leaf[label].n + count
13:    end if
14:  end for
15: end procedure

```

We do not provide the FLEX-C Centroids aggregation function as pseudocode, as it is more effectively explained using natural language due to the differential treatment of three types of leaf nodes: (1) those with a trained classifier, (2) those eligible for a classifier but left untrained due to the absence of supervised data, and (3) those too deep in the tree to allow classifiers, and thus reserved for the unsupervised "normal" class.

The FLEX-C Centroids aggregation function produces a confidence distribution over all labels ever introduced through supervision, including the unsupervised normal class. The first step, for each input sample, is to traverse all trees to collect distances from their respective classifiers, if present (as described in Algorithm 6).

We then construct an extended list of distances per tree, inserting the actual centroid

Algorithm 6 Predict Function for FLEX-C Centroids**Require:***Leaf*: The leaf classifier*x*: The input feature vector

```

1: procedure PREDICT(Leaf, x)
2:   Initialize keys  $\leftarrow []$ 
3:   Initialize distances  $\leftarrow []$ 
4:   for each label  $\in$  Leaf do
5:     keys.APPEND(label)
6:     c  $\leftarrow$  Leaf[label].centroid
7:     d  $\leftarrow$  DIST(c, x) ▷ Euclidean distance
8:     distances.APPEND(d)
9:   end for
10:  return keys, distances
11: end procedure

```

distance if the corresponding leaf contains a centroid for that label. To assign higher confidence to closer centroids, we invert the distances by multiplying them by -1 , so that smaller distances yield larger scores. These inverted distances are then passed through the softmax function:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}, \quad \text{for } i = 1, \dots, n,$$

which transforms them into a valid probability distribution over the supervised labels. This yields a confidence vector for each tree that contains at least one trained centroid. The confidence for the normal class is set to zero.

The confidence for the normal class, denoted $c[0]$, is defined as the proportion of trees whose corresponding leaf was too deep to host a classifier. Each such tree casts a full vote for the normal class. The remaining probability mass, $(1 - c[0])$, is allocated to the other labels according to the average confidence vector computed from the trees with trained classifiers. Optionally, leaves with eligible but untrained classifiers can also participate in voting. If included, each of these leaves contributes a vote equal to the average confidence vector computed from the trained classifiers. The final output confidence is obtained by averaging all contributions across trees.

Training Complexity The computational complexity of the training process can be divided into two main parts: building the unsupervised iForest structure and injecting the supervised knowledge. Let T denote the number of trees, ψ the subsampling size, S the size of the supervised dataset, F the number of features, and C the number of classes.

7.3. METHODOLOGY

1. **Isolation Forest Construction:** The initial, unsupervised phase involves two steps:

- Building the main iForest structure, which costs $O(T\psi \log \psi)$ time.
- Attaching a classifier structure to each leaf node. With T trees having up to ϕ leaves each, the computational complexity of this step is $O(T\phi)$. The reasoning for this is the same as we saw for FLEX-C Labels in Section 7.3.3.

2. **Supervised Knowledge Injection:** In this phase, we use the labeled training set of size S to add knowledge to the forest. The input data points are vectors in $\mathcal{R}^F$, so the centroids will be as well. The complexity here comes from a few operations:

- First, for each of the S data points, we have to find its corresponding leaf in each of the T trees. This tree traversal process has a total time complexity of $O(ST \log \psi)$.
- Next, we update the centroids in the leaves. The cost of this step scales linearly with S and can be broken down further:
 - We start with a single pass over the supervised data to calculate the necessary sums and counts for each class. Processing S points in an F -dimensional space takes $O(SF)$ time.
 - With these aggregates, we can apply the updates. Each leaf has a classifier with at most C centroids. Updating a single centroid is dominated by a vector addition in $\mathcal{R}^F$ (an $O(F)$ operation), as the update to the point count is a negligible $O(1)$ operation. Summing this cost over all possible centroids in the entire forest gives us a total complexity of $O(TCF)$.

Combining all these together, the total training complexity is:

$$O(T\psi \log \psi + ST \log \psi + SF + TCF)$$

In practice, this complexity is very similar to that of FLEX-C Labels. The number of features F and classes C are typically small, which means the related terms have minimal impact.

Inference Complexity We evaluate the inference complexity for a single input sample. To derive an upper bound, we assume a worst-case scenario in which every leaf node is populated with a trained classifier. The prediction computation proceeds in three steps.

1. **Finding the Leaves:** First, the input instance traverses each of the T trees to reach its corresponding leaf. This takes $O(T \log \psi)$ time, just like a standard iForest prediction.

2. **Computing Distances:** Next, we use the classifier in each leaf to get a prediction. Following the logic of Algorithm 6, we compute the distance between the input and each of the leaf's centroids (both vectors in $\mathcal{R}^F$). Since there can be up to C centroids, this step costs $O(F)$ for each one. Across all T trees, the total complexity is $O(TCF)$.
3. **Aggregating the Results:** Finally, we combine the predictions from all T trees. This involves converting each leaf's output into a confidence score for all C classes and then averaging these scores. The total cost for this aggregation is $O(TC)$. Softmax normalization factor is computed once per processed vector.

Therefore, the total cost for inference is $O(T \log \psi + TCF + TC)$. We can simplify this expression because the cost of distance computation ($O(TCF)$) is already an upper bound for the aggregation cost ($O(TC)$). Therefore, the final upper bound on inference complexity is:

$$O(T \log \psi + TCF)$$

While this appears more complex than the standard iForest, in practice the performance is often nearly identical. This is because the number of features (F) and classes (C) are typically small, meaning the runtime is still primarily driven by the $O(T \log \psi)$ term.

7.4 EXPERIMENTAL SETUP

In this section we describe the experimental setup used to evaluate our proposed models. Section 7.4.1 presents the datasets, while Sections 7.4.2 and 7.4.3 explain how we simulated the two main challenges of *nominal data contamination* and *fault data scarcity*. Section 7.4.4 introduces the baselines for comparison, Section 7.4.5 outlines our hyperparameter tuning strategy, and Section 7.4.6 defines the evaluation metrics. Finally, Section 7.4.7 provides implementation details.

7.4.1 DATA

To test our proposed models, we focused on datasets that met the following criteria:

- **Multi-class classification:** The dataset must have more than two classes to ensure a meaningful evaluation of multi-class classification performance.
- **Class imbalance:** We require a dataset with significant class imbalance, specifically containing many scarce classes to represent faults.

7.4. EXPERIMENTAL SETUP

- **Tabular format:** The dataset must be in a tabular format or already preprocessed into a such form, as our models are designed to work with structured data. Using time series or image data would require domain knowledge to extract relevant features, which is outside the scope of this work.

Based on these criteria we selected the following datasets:

Statlog (Landsat Satellite) Dataset [280, 166]: A Landsat MSS frame has four spectral bands, two in the visible region and two in the near infrared. Each pixel is stored as an eight bit integer and has spatial resolution about 80×80 meters. The full frame is 2340×3380 pixels and the sample database is extracted from a 82×100 pixel subarea for a total of 6,435 records.

Each record lists the 36 spectral values for the 3×3 block and the numeric class label associated to the central pixel. Records are provided in shuffled order so that the original image layout cannot be reconstructed. The target is one of seven land cover classes listed in Table 7.1, and there is no sample associated with class 6.

Class	Description	Samples
1	red soil	1533
2	cotton crop	703
3	gray soil	1358
4	damp gray soil	626
5	soil with vegetation stubble	707
6	mixture class (all types present)	0
7	very damp gray soil	1508

Table 7.1: Classes with meaning and number of samples in the Statlog dataset.

Glass Identification Dataset [281, 167]: This dataset consists of 214 records obtained from the chemical analysis of glass samples. Each record has 9 numerical features: refractive index, and the concentrations of oxides of sodium, magnesium, aluminum, silicon, potassium, calcium, barium, and iron. The target variable is the type of glass, classified into seven categories, as summarized in Table 7.2.

Class	Description	Samples
1	Building windows (float processed)	70
2	Building windows (non float processed)	76
3	Vehicle windows (float processed)	17
4	Vehicle windows (non float processed)	0
5	Containers	13
6	Tableware	9
7	Headlamps	20

Table 7.2: Glass types and number of samples in the Glass Identification dataset. Float processing is a glass manufacturing technique.

7.4.2 SIMULATING NOMINAL DATA CONTAMINATION

Data contamination occurs when faulty samples mistakenly appear in a dataset intended to contain only normal data. This is a common issue because the definition of *normal* can be ambiguous, and manually checking every single data point is impractical. To simulate this effect, we applied the following procedure:

1. **Balanced Supervised Set:** Construct a balanced supervised dataset ($\mathbf{x}_{\text{supervised}}$, $\mathbf{y}_{\text{supervised}}$) by identifying the smallest class and sampling the same number of instances from each class.
2. **Unsupervised Pool:** Place the remaining samples into an unsupervised dataset $\mathbf{x}_{\text{unsupervised}}$ discarding their labels.
3. **Nominal Condition Definition:** Define the nominal class 0 by merging the predominant classes within $\mathbf{x}_{\text{unsupervised}}$. This approach creates a heterogeneous *normal* distribution without the need to artificially undersample the fault classes, avoiding data waste. Crucially, this forces the model to learn a subjective definition of normality. This setup ensures our experiments align with real-world deployment scenarios, where the boundaries of a non-faulty state are often determined by domain knowledge.
4. **Contamination Injection:** Resample $\mathbf{x}_{\text{unsupervised}}$ so that 10% of its samples come from faulty data (i.e., not class 0). If the pool of normal samples is too small, faulty samples are removed; otherwise, normal samples are reduced to maintain the 10% ratio.

7.4. EXPERIMENTAL SETUP

We also relabeled the classes to a zero-based index, ensuring that the nominal class is always 0 and removing empty classes. This procedure was applied to both datasets, and after the *Nominal Condition Definition* step, the class distributions were as follows:

- **Glass Identification:** Initially $[1, 2, 3, \emptyset, 5, 6, 7]$, which became $[0, 1, 2, 3, 4, 5]$ after cleaning and rebasing. Then the building window classes (0 and 1) were merged into 0, yielding $[0, 2, 3, 4, 5]$.
- **Statlog:** Initially $[1, 2, 3, 4, 5, \emptyset, 7]$, which became $[0, 1, 2, 3, 4, 5]$ after cleaning and rebasing. Then classes 0, 2, and 5 were merged into 0, yielding $[0, 1, 3, 4]$.

7.4.3 SIMULATING FAULT DATA SCARCITY

In order to evaluate model robustness under conditions with limited labeled fault data, we designed an incremental evaluation protocol. This protocol gradually expands the training set in successive steps, which makes it possible to analyze model performance as a function of data availability. The foundation of the evaluation remains fixed throughout the entire procedure and corresponds to the unsupervised and balanced supervised datasets described in Section 7.4.2. The only additional processing consists of partitioning the supervised dataset into a training set ($\mathbf{x}_{\text{train}}, \mathbf{y}_{\text{train}}$) containing seventy percent of the samples and a testing set ($\mathbf{x}_{\text{test}}, \mathbf{y}_{\text{test}}$) containing the remaining thirty percent.

At each iteration the available training data consists of the fixed unsupervised dataset together with $(\mathbf{x}_{\text{train_iter}}, \mathbf{y}_{\text{train_iter}})$, which is extracted in a balanced manner from the supervised training portion. This dataset serves as the independent variable and increases in size at every iteration. In order to evaluate the capability of the models to categorize the *normal* class using only unsupervised information, we create an additional dataset $(\mathbf{x}_{\text{train_iter_faulty}}, \mathbf{y}_{\text{train_iter_faulty}})$ by taking $(\mathbf{x}_{\text{train_iter}}, \mathbf{y}_{\text{train_iter}})$ and removing all samples belonging to class zero.

This procedure makes it possible to measure how performance evolves as the amount of labeled fault data increases, which directly simulates the effects of fault data scarcity.

7.4.4 MODELS AND BASELINES

In our experiments we compared the proposed models with a comprehensive set of baselines designed to capture different reference points in terms of attainable performance. These baselines include an ideal upper bound, a realistic control condition, and two methods that test whether simpler uses of the available unsupervised information could already match or exceed our proposed approach.

The proposed approach is evaluated in three configurations, all trained and tested on the same datasets and following the same two stage procedure: the *fit* function is first applied to the unsupervised dataset $\mathbf{x}_{\text{unsupervised}}$ to build the iForest structure, and the *inject_knowledge* function is then applied to the supervised fault data ($\mathbf{x}_{\text{train_iter_faulty}}$, $\mathbf{y}_{\text{train_iter_faulty}}$) to train the weak classifiers. The first two configurations, FLEX-C Labels and FLEX-C Centroids, have already been described in Section 7.3.3. The third configuration, FLEX-C Centroids v2, is a variation of the centroid strategy in which leaves that received no supervised samples are still allowed to contribute to the final confidence aggregation, measuring the effect of including these structurally relevant yet untrained leaves.

Finally, our experimental comparison includes the following baseline models:

- **RF - Upper Bound Baseline:** This model serves as a practical upper bound on performance. It is a RF trained exclusively on the clean and perfectly balanced supervised dataset ($\mathbf{x}_{\text{train_iter}}$, $\mathbf{y}_{\text{train_iter}}$). Because such an ideal dataset is rarely available in real applications, this baseline represents the best case performance ceiling and is expected to consistently outperform the proposed models as well as the other baselines.
- **RF Mislead - Control:** This baseline simulates a common real-world scenario in which a standard classifier is trained directly on the available imperfect data. It consists of a RF trained on the combination of the labeled fault data ($\mathbf{x}_{\text{train_iter_faulty}}$, $\mathbf{y}_{\text{train_iter_faulty}}$) and the entire unsupervised dataset $\mathbf{x}_{\text{unsupervised}}$. All samples in $\mathbf{x}_{\text{unsupervised}}$ are explicitly assigned the *normal* label (class 0) during training.
- **Cascade:** This baseline tests whether a simpler two stage solution is effective. It is implemented as a cascade of two models. First, an iForest is trained solely on $\mathbf{x}_{\text{unsupervised}}$ to separate normal samples (class 0) from anomalous ones. Second, a RF is trained only on the supervised fault data ($\mathbf{x}_{\text{train_iter_faulty}}$, $\mathbf{y}_{\text{train_iter_faulty}}$) to classify the samples flagged as anomalous.
- **RF Naive Self:** To test whether the unsupervised information could be trivially incorporated, we adapted a self-supervised approach inspired by [271]. In the original method, unsupervised data is added iteratively if prediction confidence exceeds a threshold (90% as indicated in the original paper), but its reported complexity of $O(UTF(S+U) \log(S+U))$ (where U is the unsupervised set size) was prohibitive for our already iterative experimental setup. We therefore implemented a simplified, one shot version. This baseline is trained exactly like the upper bound model on ($\mathbf{x}_{\text{train_iter}}$, $\mathbf{y}_{\text{train_iter}}$), and then the self-supervised step is applied: We predict labels for the

7.4. EXPERIMENTAL SETUP

entire unsupervised dataset $x_{\text{unsupervised}}$ once, and retrain a new classifier on the combined supervised data and the self-labeled samples whose predicted confidence exceeds the threshold. We expect this model to:

- Outperform the upper bound if the unsupervised data is easy to exploit to improve performance.
- Underperform if the noisy self labels disrupt learning.
- Perform comparably if the approach has a neutral effect.

7.4.5 HYPERPARAMETERS TUNING

We made a deliberate decision not to perform a dedicated hyperparameter optimization process. As detailed in Section 7.4.1, the datasets suitable for our experiments were often limited in size. This constraint, combined with our experimental setup which requires multiple data partitioning beyond the simple training and testing datasets, meant that creating an additional validation dataset for hyperparameters tuning was impractical.

Consequently, we opted for models known to perform well with standard, "out-of-the-box" configurations. The choice of RFs, for example, is supported by its documented performance. In the original UC Irvine repository page, it is reported as the top-performing model for the Statlog dataset [166] and as the second-best for the Glass Identification dataset, with results closer to the top one [167]. Furthermore, a recent study [282] also identified RF as the best model for the glass classification task, albeit on a different dataset.

Standard parameters were used for both FLEX-C and iForest, with ensembles of 100 trees and bootstrapping enabled (`bootstrap=true`). The iForest `contamination` parameter, specifying the expected proportion of anomalies and guiding the model's internal decision threshold, was set to 10%.

The only models that would have required specific tuning of their `margin` hyperparameter were our proposed FLEX-C variants. This is because their per-tree voting mechanism, while sharing similarities with iForest, is fundamentally different. After consideration, we made the following decisions:

- In **FLEX-C Labels**, we opted not to modify the underlying iForest threshold. This decision was based on its closer resemblance to the original ALIF model and the fact that its classifiers are inherently weaker.
- In **FLEX-C Centroids**, having more robust classifiers, we decided to rely more on their classification power. We achieved this by lowering its decision threshold by setting a

margin of 0.1.

As the results in Section 7.5 will show, this direct approach was sufficient to obtain a definitive positive answer to our research question. We acknowledge, however, that more extensive testing is needed, and we will propose this point in Section 7.6.

7.4.6 EVALUATION METRICS

To evaluate the performance of our classification models, it is necessary to select a suitable metric. We have selected **accuracy** as our primary metric due to its straightforward interpretation. Accuracy represents the proportion of all instances that the model has classified correctly.

The calculation of accuracy for a multi-class problem with C classes is based on the confusion matrix. A confusion matrix is a $C \times C$ table that summarizes classification performance by showing the relationship between the true and predicted classes. The fundamental components for the metric are defined for each class i as follows:

- **True Positives for class i (TP_i):** The number of instances of class i correctly classified as class i .
- **True Negatives for class i (TN_i):** The number of instances not belonging to class i correctly classified as not belonging to class i .
- **False Positives for class i (FP_i):** The number of instances not belonging to class i that were incorrectly classified as class i .
- **False Negatives for class i (FN_i):** The number of instances of class i incorrectly classified as a different class.

The formula for overall accuracy is the sum of correct predictions for all classes divided by the total number of instances:

$$\text{Accuracy} = \frac{\sum_{i=1}^C TP_i}{\text{Total Number of Instances}} \quad (7.6)$$

It is widely acknowledged that accuracy has significant limitations, particularly when used with imbalanced test sets. In cases where some classes are much more frequent than others, a model can achieve a high accuracy score simply by predicting the majority class for every instance. This gives a deceptive impression of high performance, while the model may completely fail to identify instances from minority classes.

7.5. RESULTS AND DISCUSSION

However, this limitation is not a concern for our study. As detailed in our experimental design, all our test sets are perfectly balanced, with an equal number of instances for each class. This approach ensures that accuracy provides a fair and reliable measure of performance. Therefore, a high score directly reflects a model’s ability to perform well across all classes, without the risk of being skewed by class imbalance.

While accuracy provides a valuable overall performance score, a deeper, qualitative understanding of model behavior is also necessary. To identify specific error patterns, such as which classes are commonly confused for one another, we will visualize and analyze select confusion matrices in the results chapter (Section 7.5). This analysis will allow for a more detailed interpretation of our models’ predictive behaviors.

7.4.7 IMPLEMENTATION DETAILS

All experiments were carried out in Python using `scikit-learn` [259] for the implementations of iForest and RFs, as well as for computing all performance metrics. The proposed FLEX-C variants were implemented as classes inheriting directly from `scikit-learn`’s iForest and employed `NumPy` [260] for efficient numerical operations. The implementation was released as open-source¹

Experiments were executed on a machine equipped with an AMD Ryzen Threadripper 1920X 12-core processor and 128 GB of DDR4 2133 MHz RAM.

7.5 RESULTS AND DISCUSSION

This section presents the empirical results of the experiments detailed in Section 7.4. The goal of our evaluation is twofold. First, we aim to quantify the performance degradation of a standard ML classifier, in this case a RF, when faced with data challenges common to industrial settings. These challenges include label noise in the nominal class data and a severe imbalance caused by the scarcity of fault samples. Second, to assess the effectiveness of our proposed models, FLEX-C Labels and FLEX-C Centroids, which were specifically designed for robustness against these conditions.

In the remainder, we first report the raw results and then provide a more in-depth analysis for each of the two datasets.

¹FLEX-C implementation is available at <https://github.com/AMCO-UniPD/FLEX-C/tree/publication>.

7.5.1 GLASS IDENTIFICATION

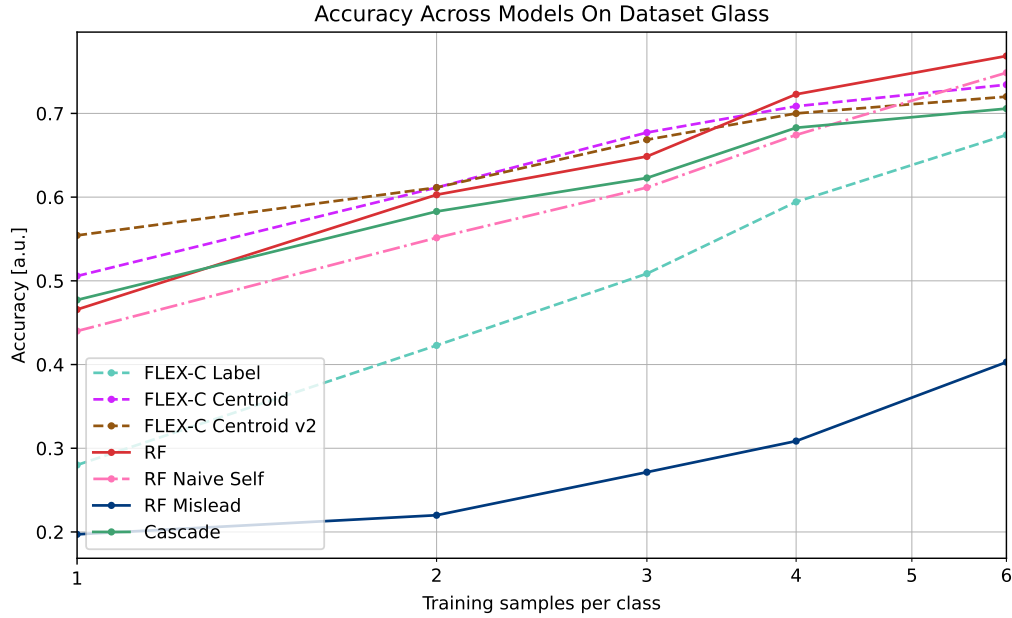


Figure 7.1: Accuracy performance of models on the Glass Identification dataset. The plot is presented on a logarithmic scale to facilitate interpretation.

The Glass Identification dataset is representative of scenarios characterized by a low to medium-low amount of labeled data, a range of significant practical interest. We therefore use this dataset to introduce our initial analysis and observations. These findings will then be extended and confirmed through the analysis of the Landsat Satellite dataset, which also covers the scenarios with more labeled data available.

Figure 7.1 highlights the strong negative impact of the scenario’s challenges on model performance. The original dataset contains seven classes, which after the preprocessing described in Section 7.4.2 were reduced to five: the system’s nominal condition and four fault classes. Consequently, in our balanced test set, an accuracy of 0.2 corresponds to random guessing. This behavior is observed for the RF_Mislead model when trained with very few faulty samples, indicating that it failed to learn any meaningful patterns.

It is also evident that RF_Naive_self performs slightly worse than the upper baseline RF, suggesting that the self-supervised labels introduced more noise than useful information, which slightly reduced the model’s performance.

The FLEX-C Labels model initially performs close to random chance when only one labeled Samples Per Fault (SPF) is available, but it rapidly improves as more data is added. To interpret this behavior, we consider the Cascade model, in which the RF component is trained on the same data as the upper baseline RF (with the normal class removed). The

7.5. RESULTS AND DISCUSSION

strong performance of the Cascade model indicates that the underlying iForest structure, which is shared with FLEX-C Labels, is effective at modeling the normal class. Consequently, the initial low performance of FLEX-C Labels must stem from too few of its internal weak classifiers having received supervised training.

Switching to FLEX-C Centroids, we can conclude that increasing the number of leaves that can receive supervised training via the *margin* parameter, along with its more advanced aggregation mechanism, improves the model ability recognize the faulty classes. The confusion matrices in Tables 7.3a and 7.3b confirm this intuition. While some misclassifications of fault classes are expected due to very few supervised samples, the nature of the errors is radically different. While FLEX-C Labels almost exclusively predicts the normal class, FLEX-C Centroids is more effective, only confusing class 1 with the normal class (a plausible error, as both represent types of glass used in window manufacturing).

Finally, as confirmed from the analyses on the second dataset, another pattern emerges: FLEX-C Centroids v2 performs better than FLEX-C Centroids with very few samples but starts to fall behind as more labeled data is used. We will look at this behavior more closely in the Landsat Satellite dataset analysis.

Table 7.3: Average confusion matrices across all seeds for the FLEX-C Labels and FLEX-C Centroids models on Glass Identification dataset with one SPF. The diagonal cells (green) represent correct classifications, while the off-diagonal cells (red) represent misclassifications.

		Predicted				
		0	2	3	4	5
True	0	2.36	0.32	0.00	0.00	0.00
	2	2.28	0.56	0.00	0.00	0.00
	3	2.56	0.00	0.20	0.00	0.00
	4	2.48	0.00	0.04	0.36	0.00
	5	2.4	0.00	0.00	0.00	0.44
(a) FLEX-C Labels						
		Predicted				
		0	2	3	4	5
True	0	2.60	0.00	0.08	0.00	0.00
	2	2.80	0.04	0.00	0.00	0.00
	3	1.00	0.00	1.08	0.20	0.48
	4	0.76	0.04	0.28	1.52	0.28
	5	0.64	0.00	0.12	0.24	1.84
(b) FLEX-C Centroids						

7.5.2 LANDSAT SATELLITE

The Landsat Satellite dataset offers a stronger basis for our final analysis than the Glass Identification dataset because it contains a much larger number of samples. We rely on the results from this dataset to determine whether our research question has been fully addressed and to verify that the proposed models behave as expected. In contrast, the Glass

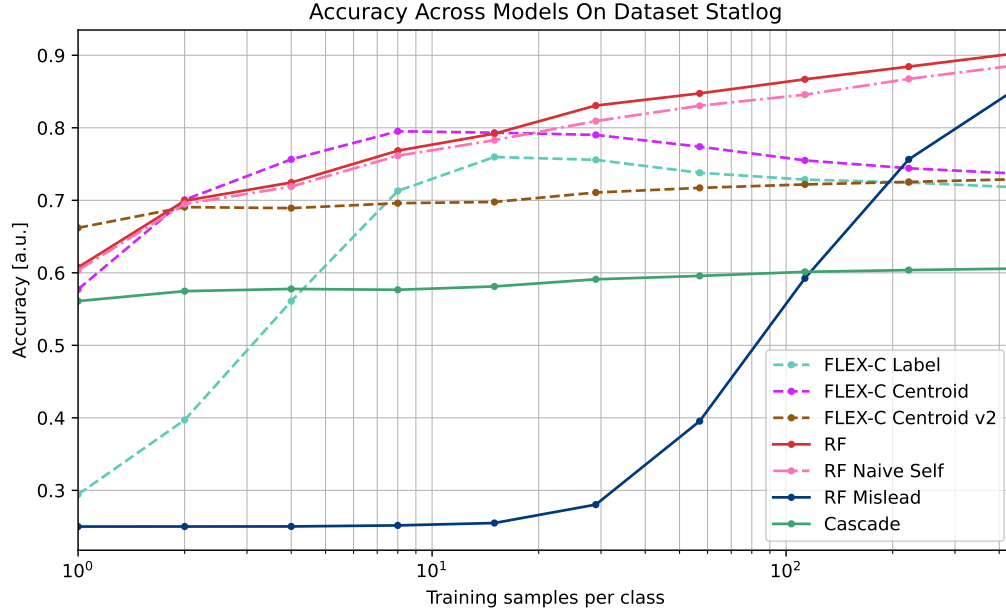


Figure 7.2: Accuracy performance of models on the Landsat Satellite dataset. The plot is presented on a logarithmic scale to facilitate interpretation.

Identification dataset, after preprocessing in our experimental setup, yielded only three test samples per class. Although we applied a testing protocol with fifty runs using different random seeds, similar to k-fold cross-validation to reduce variance between observed and true performance, the limited size makes it less reliable for drawing definitive conclusions.

As shown in Figure 7.2, our experiments on the Landsat dataset utilize hundreds of supervised points per fault (438 to be precise, with 188 test samples per class). The original dataset contains seven classes, which after the preprocessing described in Section 7.4.2 were reduced to four. Consequently, the accuracy threshold for random chance is 0.25.

First and foremost, the results on this dataset make it definitively clear that the combined challenges of nominal data contamination and fault data scarcity render a common classifier unable to learn effectively. The accuracy of RF_Mislead, shown in Figure 7.2, is on par with random choice for up to 29 supervised Samples Per Fault (SPF) and remains considerably below the other models even at 113 SPF. It is worth noting that acquiring hundreds of labeled samples for a single fault category pushes the boundaries of what is feasible in most industrial realities. This challenge is compounded by the fact that these samples must be representative of the entire fault category, not just consecutive sensors' points from a single fault event.

First and foremost, the results on this dataset clearly demonstrate that the combined challenges of nominal data contamination and fault data scarcity prevent a standard classifier

7.5. RESULTS AND DISCUSSION

from learning effectively. The accuracy of RF_Mislead, shown in Figure 7.2, remains at the level of random guessing for up to 29 supervised Samples Per Fault (SPF) and stays well below the other models even at 113 SPF. It is important to note that obtaining hundreds of labeled samples for a single fault category is often infeasible in most industrial settings, considering these samples must represent the entire fault category, rather than consecutive sensor readings from a single fault event.

The performance trajectory of FLEX-C Labels reinforces our earlier observation: training a sufficient number of weak classifiers is challenging when the supervised dataset is small. Nevertheless, with as few as 8 SPF, it becomes the fourth-best model, trailing only the upper bound RF, the RF_Naive_Self, and FLEX-C Centroids.

The results of the Cascade model provide clear evidence that robustness in this scenario cannot be achieved by naively combining supervised and unsupervised classifiers, even if these classifiers perform well individually under less demanding conditions.

The RF_Naive_self model begins with performance identical to the upper bound RF. However, as more supervised training samples are introduced, the performance gap between them widens. This demonstrates that incorporating unsupervised information into a model's knowledge is not a trivial task; our proposed models achieve this natively through their iForest structure. The performance gap widens because the noise introduced by this ambiguous unsupervised information becomes more disruptive as the underlying supervised model's performance improves.

The RF_Naive_Self model initially performs at the same level as the upper bound RF. However, as additional supervised training samples are provided, the performance gap between them increases. This illustrates that incorporating unsupervised information into a model, which our models achieve naturally through their iForest structure, is not a trivial task. The widening gap occurs because the noise introduced by the ambiguous unsupervised data becomes increasingly disruptive as the underlying supervised model improves.

Finally, the performance of FLEX-C Centroids largely aligns with that of the upper bound baseline, and it even slightly surpasses it around 10 SPF. Considering that our model is an implementation of our general concept relying on simple classifiers and without extensive hyperparameter tuning, achieving performance comparable to a SOTA model on both datasets validates the soundness of our proposed approach.

Analyzing Figure 7.2 more closely reveals a few noteworthy behaviors of the FLEX-C models:

- Consistent with the results on the Glass Identification dataset, allowing non-trained weak classifiers to participate in the prediction vote (i.e., the v2 variant of FLEX-C

Centroids) improves performance at very low SPF, but significantly degrades it as SPF increases.

- All FLEX-C models converge to a fixed accuracy once a sufficiently high SPF is reached, approaching this plateau from either below or above.

To explain these phenomena, we examine the confusion matrices for a single SPF, presented in Tables 7.4a and 7.4b, as well as for the maximum SPF, shown in Tables 7.5a and 7.5b.

With one SPF, the model is overly inclined to classify data as not anomalous (class 0). In this context, accounting for the vote of untrained classifiers counteracts this bias towards the normal class, effectively improving the model's overall performance.

Conversely, when the majority of weak classifiers have been trained, the model is already effective at identifying anomalous data. The additional votes from the few remaining untrained classifiers push the model even further in that direction, exacerbating the model's tendency to predict faults and thereby limiting its performance. This analysis points to a straightforward potential improvement: the vote from untrained classifiers could be used as a dynamic balancing factor. When very few classifiers are trained, an untrained classifier is more likely to reside in a leaf node that could contain faulty data, and its vote should thus favor the faulty side. In contrast, when many classifiers are already trained, an untrained classifier is more likely in a leaf that, for sampling or splitting reasons, appears potentially anomalous but likely contains normal data; its vote should therefore favor the normal class. However, a thorough examination and implementation of this balancing mechanism is left for future research.

In Section 7.3, we hypothesized that the coarse tree partitioning from the iForest might act as a strong regularizer, limiting the model's performance. If that were true, we would expect to see a high number of misclassifications between different fault types. Looking at the confusion matrices, it is clear that while these "wrong fault" errors certainly exist, they are not the main reason for the accuracy gap compared to the upper bound RF. Instead, the primary component of this performance difference appears to be the role of untrained weak classifiers in the prediction aggregation.

7.5.3 COMPUTATIONAL COMPLEXITY

This section summarizes and discusses the computational complexity of the models and baselines considered in this work, with reference to the detailed breakdown in Table 7.6. The analysis is presented separately for the training and inference phases.

7.5. RESULTS AND DISCUSSION

Table 7.4: Average confusion matrices across all seeds for the FLEX-C Centroids and FLEX-C Centroids v2 models on Landsat Satellite dataset with 1 SPF. The diagonal cells (green) represent correct classifications, while the off-diagonal cells (red) represent misclassifications.

		Predicted			
		0	1	3	4
True	0	179.84	0.04	3.8	4.32
	1	29.24	148.76	0.44	9.56
	3	151.44	0.24	36.0	0.32
	4	114.92	2.76	0.88	69.44

(a) FLEX-C Centroids

		Predicted			
		0	1	3	4
True	0	57.2	16.64	71.96	42.2
	1	7.08	159.36	5.24	16.32
	3	19.0	5.6	148.28	15.12
	4	11.28	21.32	22.4	133.0

(b) FLEX-C Centroids v2

Table 7.5: Average confusion matrices across all seeds for the FLEX-C Centroids and FLEX-C Centroids v2 models on Landsat Satellite with 438 SPF. The diagonal cells (green) represent correct classifications, while the off-diagonal cells (red) represent misclassifications.

		Predicted			
		0	1	3	4
True	0	9.32	0.76	98.04	79.88
	1	0.20	181.6	2.56	3.64
	3	0.32	1.08	183.04	3.56
	4	0.56	2.16	5.04	180.24

(a) FLEX-C Centroids

		Predicted			
		0	1	3	4
True	0	3.24	0.76	98.88	85.12
	1	0.16	181.6	2.56	3.68
	3	0.2	1.08	183.12	3.6
	4	0.56	2.16	5.04	180.24

(b) FLEX-C Centroids v2

TRAINING COMPLEXITY

During training, the models exhibit markedly different computational demands.

Random Forest (RF)-based models, used as baselines, have the highest training complexity, $O(T \cdot F \cdot S \log S)$, where the dominant factor is the log-linear dependence on the total number of supervised training samples S .

In contrast, the proposed FLEX-C models leverage the efficiency of the iForest structure to reduce the complexity associated with S to a linear relationship.

Specifically, FLEX-C Centroids has a training complexity of $O(T\psi \log \psi + ST \log \psi + SF + TCF)$, where T is the number of trees, ψ the unsupervised data subsample size, C the number of classes, and F the number of features. Although several terms contribute to this complexity, the dominant factor in practical scenarios is the supervised knowledge injection,

which scales linearly with both the number of trees T and the number of supervised samples S .

Finally, the FLEX-C Labels variant exhibits a slightly lower complexity, $O(T\psi \log \psi + ST \log \psi)$, since it omits the terms dependent on C and F that arise from centroid operations. This reduction makes it the fastest model among the proposed methods during training.

INFERENCE COMPLEXITY

Regarding inference complexity, FLEX-C Labels is the most efficient model, with a complexity of $O(T \log \psi)$. Notably, it introduces no additional asymptotic overhead beyond that of the underlying iForest structure.

The RF-based baselines follow, with an inference complexity of $O(T \log S)$, where performance is mainly determined by the depth of each tree, which grows logarithmically with the number of supervised training samples.

The most computationally demanding model during inference is FLEX-C Centroids, with a complexity of $O(T \log \psi + TCF)$. The added cost originates from the distance computations performed by the leaf-level classifiers, which scale with the number of trees, classes, and features.

Table 7.6: Comparison of theoretical time complexities for training and inference across different models. The variables used are defined below.

Model	Training Complexity	Inference Complexity (per sample)
RF	$O(TSK \log S)$	$O(T \log S)$
FLEX-C Labels	$O(T\psi \log \psi + ST \log \psi)$	$O(T \log \psi)$
FLEX-C Centroids	$O(T\psi \log \psi + ST \log \psi + SF + TCF)$	$O(T \log \psi + TCF)$

Variables Definitions:

T : The number of trees in the forest.

K : The subset of features considered by RF.

ψ : The subsampling size used to train FLEX-Cs trees using unsupervised data.

S : The number of supervised samples.

F : The number of features (dimensionality) of the data.

C : The total number of classes.

7.6 SUMMARY OF FINDINGS

In this chapter, we analyzed the impact of two critical challenges in industrial fault identification: *Nominal Data Contamination* and *Fault Data Scarcity*. These issues arise from a common industrial reality where raw, unlabeled data characterizing the nominal working condition of a system is abundant, yet the cost of thoroughly validating this data means it is often contaminated with unlabeled faulty points. Concurrently, since systems are designed to operate correctly most of the time, officially recorded and labeled fault instances are rare, leading to very small training datasets for each fault category.

Our primary contribution was to develop an experimental setup that effectively reproduces these challenges. Using this setup, we first demonstrated that a SOTA classifier, RF, struggles to learn effectively under these conditions, confirming the severity of the problem. In response, we proposed a new family of models, the FLEX-C, with two specific implementations: FLEX-C Labels and FLEX-C Centroids. These models were specifically designed to learn robustly in this challenging environment. We then tested our models against RF and other relevant baselines to validate their performance.

The results of our experiments led to several key conclusions. Our results show that the combination of contaminated normal data and scarce fault samples prevents a standard RF from learning effectively. This limitation persists across the entire range of training set sizes realistically achievable in an industrial setting. In stark contrast, our proposed FLEX-C models, despite facing the same challenges, achieved performance comparable to a RF trained *without* these challenges. In fact, our models even outperformed this baseline in the range of approximately ten supervised samples per fault.

Furthermore, the performance of a RF trained with a simple self-supervised strategy demonstrates the inherent difficulty of exploiting unsupervised nominal data—a challenge our models overcome by natively integrating this information through their iForest structure. The soundness of our general idea is further reinforced by the fact that a cascade of an iForest and a RF was almost never superior to our integrated FLEX-C models. Finally, from a practical standpoint, our theoretical time complexity analysis shows that our models are not significantly more computationally expensive than a standard RF.

It is important, however, to acknowledge the limitations of this work. The FLEX-C models' hyperparameters were not tuned, as this would have required a larger dataset. Additionally, our implementations are research prototypes developed in Python, and their real-time performance is therefore not directly comparable to the highly optimized, production-grade libraries available for RF and iForest. Lastly, we recognize that the role of non-trained weak classifiers during the prediction phase of the FLEX-C models is an area with clear potential

for improvement.

7.6.1 DIRECTIONS FOR FUTURE RESEARCH

While this work successfully achieved its primary goals, the performance of our proposed methods exceeded expectations, opening several promising avenues for future research to explore their full potential.

One primary direction is to enhance the role of the non-trained weak classifiers. As noted in our analysis, their current contribution to the voting process is static. Future work could explore extending these classifiers into dynamic balancing agents that adapt their voting weights according to the training state of the forest. Such an approach would enable the model to counteract its natural bias toward the normal class when only a few classifiers have received supervised training, and likewise mitigate the opposite bias toward the fault classes as more classifiers become available.

A second crucial research path involves a rigorous procedure for tuning the model's hyperparameters, particularly the *margin* that governs which leaves are eligible to host classifiers. Proper tuning with a dedicated validation dataset would not only allow the models to reach their optimal performance but also enable testing our approach on a wider variety of datasets, generalizing our findings.

A third area of extension would be to explore the use of more complex weak classifiers within the leaves. While this work focused on simple labels and centroids to manage data scarcity, scenarios with slightly more available data could benefit from more sophisticated local experts, potentially further improving the model's discriminatory power.

Finally, a more advanced research direction is to adapt the unsupervised structure of the forest so that it better reflects user defined notions of normality. In our experiments, the normal class was formed by aggregating several original dataset classes, yet the forest structure remained agnostic to this semantic grouping. Improving the alignment between the ensemble and such user specific definitions could be achieved by integrating mechanisms inspired by the previously discussed ALIF and BALIF. These approaches offer a principled way to adjust leaf depths through active user feedback, thereby enabling or disabling expert classifiers in targeted regions of the feature space. Incorporating similar ideas into our framework would allow the forest to reshape its unsupervised structure to better capture domain knowledge.



Conclusions and Future Directions

In this work we have investigated key challenges in deploying data-driven predictive maintenance approaches in real-world industrial scenarios. We proposed and validated novel methodologies and algorithms that form a toolbox of techniques designed for efficiency, robustness, and seamless integration with human expertise, aligning with the human-in-the-loop, sustainability, and resilience principles of Industry 5.0. Our approach combined proxy tasks that translate hydroelectric predictive maintenance scenarios into general machine learning problems, ensuring domain focus, broader applicability, and reproducibility.

In this final chapter, we summarize the main contributions of the thesis, highlighting the limitations of our approaches and suggesting possible directions for future research.

8.1 SUMMARY OF CONTRIBUTIONS

The main contributions of this thesis span three complementary directions, with four projects addressing efficiency, robustness, and human centrality in PdM.

In Chapter 4, we proposed a methodology that combines anomaly detection with SHAP-based interpretability to support fault detection and root cause analysis in vibration monitoring for hydropower plants. Using real data from hydroelectric facilities, we showed that Autoencoders offer the best balance between performance, accuracy, and output readability, while SHAP is effective in highlighting to the expert the features that simplify the fault identification process. This promotes *human centrality* by providing interpretable results that empower experts, foster trust, and ensure accountability in AI-driven solutions.

In Chapter 5, we explored driver identification as a proxy task for industrial fault iden-

8.1. SUMMARY OF CONTRIBUTIONS

tification, where recognizing different drivers maps directly to recognizing different fault types. This formulation allowed us to study how a model can efficiently learn to recognize new classes as they appear, mirroring the introduction of new faults in an industrial setting.

The central aim of this chapter was to investigate how continual learning can make this class expansion process computationally efficient, enabling deployment on resource constrained edge devices rather than relying on cloud based infrastructure. To this end, we designed three progressively challenging continual learning scenarios on the OCSLab dataset to approximate real operating conditions and evaluated state-of-the-art methods under these constraints.

We also introduced two new approaches, SmooER and SmooDER, which exploit the natural temporal continuity present in both driver identity and fault evolution. These methods consistently outperform existing techniques and demonstrate that continual learning can improve efficiency by reducing the computational burden, making real time and edge compatible driver identification more feasible. In Chapter 5, we explored driver identification as a proxy task for industrial fault identification, where recognizing different drivers maps directly to recognizing different fault types. This formulation allowed us to study how a model can efficiently learn to recognize new classes as they appear, mirroring the introduction of new faults in an industrial setting.

The central aim of this chapter was to investigate how continual learning can make this class expansion process computationally efficient, enabling deployment on resource constrained edge devices rather than relying on cloud based infrastructure. To this end, we designed three progressively challenging continual learning scenarios on the OCSLab dataset to approximate real operating conditions and evaluated state-of-the-art methods under these constraints.

We also introduced two new approaches, SmooER and SmooDER, which exploit the natural temporal continuity present in both driver identity and fault evolution. These methods consistently outperform existing techniques and demonstrate that continual learning can improve efficiency by reducing the computational burden, making real time and edge compatible driver identification more feasible.

Chapter 6 introduced RootIF, a feature-evolving anomaly detection method that reduces data and computational requirements when retraining models as new features or sensors are added. This approach enables faster deployment without historical data, while improving privacy and reducing vulnerability windows. We compared it to xStream and found competitive or superior performance across several public datasets. Chapter 6 introduced RootIF, a feature-evolving anomaly detection method that reduces data and computational requirements when retraining models as new features or sensors are added. This approach enables

faster deployment without historical data, while improving privacy and reducing vulnerability windows. We compared it to xStream and found competitive or superior performance across several public datasets.

Finally, in Chapter 7, we addressed fault detection and identification under the combined challenges of contaminated nominal data and scarce fault data. To this end, we proposed FLEX-C, a family of semi-supervised structured ensemble models, and developed two variants: FLEX-C Labels and FLEX-C Centroids. Through systematic comparison with Random Forest and other baselines, we demonstrated that FLEX-C is inherently robust to these conditions and can match or surpass the performance of a Random Forest trained on clean and balanced data within the range of realistically obtainable dataset sizes.

8.2 LIMITATIONS AND FUTURE DIRECTIONS

While the contributions of this thesis advance the state-of-the-art, several limitations and open questions remain, paving the way for future research. Considering the four projects in order of presentation, we progress from applying existing methods to a specific context in Chapter 4, to incremental improvements of established approaches in Chapter 5, to the proposal of a new method in Chapter 6, and finally the development of an entirely new framework in Chapter 7 inspired by existing literature. This trajectory reflects both the increasing complexity and novelty of the challenges addressed, as well as the corresponding growth in limitations and opportunities for further exploration.

The main limitation of Chapter 4 is the computational cost of SHAP, which can hinder real-time applications. Future work could investigate more efficient interpretability techniques. In Chapter 5, continual learning methods remain vulnerable to data poisoning attacks, a critical issue for driver identification and industrial contexts alike. Future research should focus on approaches that balance adaptability with robustness.

Chapter 6 showed that our proposed method can reach state-of-the-art performance in feature-evolving anomaly detection. Its current implementation, however, offers several opportunities for further development. The method could be optimized in terms of computational efficiency and memory usage, extended from an iterative to a fully online learning framework, and enriched with complementary interpretability techniques to refine the weighting mechanism used within the ensemble.

Finally, FLEX-C in Chapter 7 has shown promising results, but current implementations remain proof of concept and only scratch the surface of the framework’s potential. We deliberately excluded an active learning component to keep the focus on validating applicability

8.2. LIMITATIONS AND FUTURE DIRECTIONS

and effectiveness, leaving future work to explore richer implementations, including online and active learning settings, to evaluate the approach in more diverse scenarios. Finally, FLEX-C in Chapter 7 has shown promising results, but current implementations remain proof of concept and only scratch the surface of the framework’s potential. We deliberately excluded an active learning component to keep the focus on validating applicability and effectiveness, leaving future work to explore richer implementations, including online and active learning settings, to evaluate the approach in more diverse scenarios.

Bibliography

- [1] Henning Kagermann, Wolfgang Wahlster, and Johannes Helbig. *Recommendations for implementing the strategic initiative Industrie 4.0: Final report of the Industrie 4.0 Working Group*. Tech. rep. Published by Acatech (National Academy of Science and Engineering). Forschungsunion, Apr. 2013. URL: <https://en.acatech.de/publication/recommendations-for-implementing-the-strategic-initiative-industrie-4-0-final-report-of-the-industrie-4-0-working-group/> (visited on 08/08/2025).
- [2] Yongkui Liu and Xun Xu. “Industry 4.0 and Cloud Manufacturing: A Comparative Analysis”. In: *Journal of Manufacturing Science and Engineering* 139.3 (Oct. 2016). ISSN: 1087-1357. DOI: 10.1115/1.4034667. eprint: https://asmedigitalcollection.asme.org/manufacturingscience/article-pdf/139/3/034701/6273117/manu_139_03_034701.pdf.
- [3] European Commission, Directorate-General for Research, Innovation, M. Breque, L. De Nul, and A. Petridis. *Industry 5.0 - Towards a sustainable, human-centric and resilient European industry*. Publications Office of the European Union, 2021. DOI: doi/10.2777/308407.
- [4] Mariia Golovianko, Vagan Terziyan, Vladyslav Branytskyi, and Diana Malyk. “Industry 4.0 vs. Industry 5.0: Co-existence, Transition, or a Hybrid”. In: *Procedia Computer Science* 217 (2023). 4th International Conference on Industry 4.0 and Smart Manufacturing, pp. 102–113. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2022.12.206>.
- [5] IBM. *What is Predictive Maintenance?* IBM Topics. May 2023. URL: <https://www.ibm.com/topics/predictive-maintenance> (visited on 08/08/2025).
- [6] Eurostat. *Shedding light on energy in Europe - 2025 edition*. Interactive digital publication. Mar. 2025. URL: <https://ec.europa.eu/eurostat/web/interactive-publications/energy> (visited on 08/08/2025).

BIBLIOGRAPHY

- [7] Z. Glasnovic and K. Margeta. "Chapter 8 - Revitalization of Hydro Energy: A New Approach for Storing Solar Energy". In: *Solar Energy Storage*. Ed. by Bent Sørensen. Boston: Academic Press, 2015, pp. 189–206. ISBN: 978-0-12-409540-3. DOI: <https://doi.org/10.1016/B978-0-12-409540-3.00008-6>.
- [8] International Energy Agency. *Energy Security*. International Energy Agency (IEA). 2025. URL: <https://www.iea.org/topics/energy-security> (visited on 08/08/2025).
- [9] University of Plymouth. *What is energy security?* University of Plymouth. 2025. URL: <https://www.plymouth.ac.uk/discover/what-is-energy-security> (visited on 08/08/2025).
- [10] Lorenzo Brigato and Luca Iocchi. "A Close Look at Deep Learning with Small Data". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 2490–2497. DOI: 10.1109/ICPR48806.2021.9412492.
- [11] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. "Learning with noisy labels". In: *Advances in neural information processing systems* 26 (2013).
- [12] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.
- [13] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. "A continual learning survey: Defying forgetting in classification tasks". In: *IEEE transactions on pattern analysis and machine intelligence* 44.7 (2021), pp. 3366–3385.
- [14] Mohammad M. Masud, Qing Chen, Latifur Khan, Charu C. Aggarwal, Jing Gao, Jiawei Han, Ashok Srivastava, and Nikunj C. Oza. "Classification and Adaptive Novel Class Detection of Feature-Evolving Data Streams". In: *IEEE Transactions on Knowledge and Data Engineering* 25.7 (2013), pp. 1484–1497. DOI: 10.1109/TKDE.2012.109.
- [15] European Parliament and the Council of the European Union. *Article 14 - Human Oversight*. artificialintelligenceact.eu. 2024. URL: <https://artificialintelligenceact.eu/article/14/> (visited on 08/08/2025).
- [16] Aysegul Ucar, Mehmet Karakose, and Necim Kırımça. "Artificial Intelligence for Predictive Maintenance Applications: Key Components, Trustworthiness, and Future Trends". In: *Applied Sciences* 14.2 (2024). ISSN: 2076-3417. DOI: 10.3390/app14020898.

- [17] Nuuraan Risqi Amaliah, Benny Tjahjono, and Vasile Palade. "Human-in-the-Loop XAI for Predictive Maintenance: A Systematic Review of Interactive Systems and Their Effectiveness in Maintenance Decision-Making". In: *Electronics* 14.17 (2025). ISSN: 2079-9292. DOI: 10.3390/electronics14173384.
- [18] Mattia Fanan, Claudio Baron, Ruggero Carli, Marc-Aurèle Divernois, Jean-Christophe Marongiu, and Gian Antonio Susto. "Anomaly Detection for Hydroelectric Power Plants: a Machine Learning-based Approach". In: *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*. 2023, pp. 1–6. DOI: 10.1109/INDIN51400.2023.10218027.
- [19] Mattia Fanan, Davide Dalle Pezze, Emad Efatinasab, Ruggero Carli, Mirco Rampazzo, and Gian Antonio Susto. "Continual Learning for Behavior-based Driver Identification". In: *Engineering Applications of Artificial Intelligence* 150 (2025), p. 110459. ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2025.110459>.
- [20] Mattia Fanan, Valentina Zaccaria, Enxhi Nushi, Ruggero Carli, and Gian Antonio Susto. "Feature-Evolving Anomaly Detection via Root-forced Isolation Forest". In: *Engineering Applications of Artificial Intelligence* (2025). Manuscript submitted for publication.
- [21] Mattia Fanan, İclem Naz Bakirci, Ruggero Carli, and Gian Antonio Susto. "FLEX-C: A Robust Framework for Fault Detection and Identification under Scarce and Contaminated Data". In: *Engineering Applications of Artificial Intelligence* (2025). Manuscript submitted for publication.
- [22] Tommaso Barbariol, Davide Masiero, Mattia Fanan, Enrico Feltresi, and Gian Antonio Susto. "Time Series Forecasting to Detect Anomalous Behavior in Multiphase Flow Meters". In: *2024 IEEE 8th Forum on Research and Technologies for Society and Industry Innovation (RTSI)*. 2024, pp. 54–59. DOI: 10.1109/RTSI61910.2024.10761679.
- [23] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.
- [24] Ashwani Sharma and Bikram Jit Singh. "Evolution of industrial revolutions: A review". In: *International Journal of Innovative Technology and Exploring Engineering* 9.11 (2020), pp. 66–73.
- [25] Mario Rupp, Max Schneckenburger, Markus Merkel, Rainer Börret, and David K. Harrison. "Industry 4.0: A Technological-Oriented Definition Based on Bibliometric Analysis and Literature Review". In: *Journal of Open Innovation: Technology, Market, and Complexity* 7.1 (2021). ISSN: 2199-8531. DOI: 10.3390/joitmc7010068.

BIBLIOGRAPHY

- [26] Pradyumna Gokhale, Omkar Bhat, and Sagar Bhat. "Introduction to IOT". In: *International Advanced Research Journal in Science, Engineering and Technology* 5.1 (2018), pp. 41–44.
- [27] Samuel Fosso Wamba, Shahriar Akter, Andrew Edwards, Geoffrey Chopin, and Denis Gnanzou. "How 'big data' can make big impact: Findings from a systematic review and a longitudinal case study". In: *International journal of production economics* 165 (2015), pp. 234–246.
- [28] Goran Putnik, Luis Ferreira, Nuno Lopes, and Zlata Putnik. "What is a Cyber-Physical System: Definitions and models spectrum". In: *FME Transactions* 47 (Jan. 2019), pp. 663–674. doi: 10.5937/fmet1904663P.
- [29] Hybrid Cloud. "The nist definition of cloud computing". In: *National institute of science and technology, special publication* 800.2011 (2011), p. 145.
- [30] Manambedu Vijayakumar Raja, Himanshu Thaker, Subba rao Katragadda, and Supriya Kadam. "A Review of Human-Centric AI in Industry 5.0: Integrating Data Science with Mechanical Automation". In: *Journal of Economics, Finance and Accounting Studies* 7.4 (2025), pp. 42–53.
- [31] Paula Fraga-Lamas, Sérgio Ivan Lopes, and Tiago M Fernández-Caramés. "Green IoT and edge AI as key technological enablers for a sustainable digital transition towards a smart circular economy: An industry 5.0 use case". In: *Sensors* 21.17 (2021), p. 5745.
- [32] Balasaheb S Gandhare and Milind Akarte. "Maintenance strategy selection". In: *Ninth AIMS International Conference on Management*. AIMS International Maharashtra, India. 2012, pp. 1330–1336.
- [33] M Bevilacqua and M Braglia. "The analytic hierarchy process applied to maintenance strategy selection". In: *Reliability Engineering & System Safety* 70.1 (2000), pp. 71–83. ISSN: 0951-8320. doi: [https://doi.org/10.1016/S0951-8320\(00\)00047-8](https://doi.org/10.1016/S0951-8320(00)00047-8).
- [34] Evan Zabawski. *Maintenance nomenclature*. Park Ridge, IL : Dec. 2020.
- [35] Geert Waeyenbergh and Liliane Pintelon. "A framework for maintenance concept development". In: *International Journal of Production Economics - INT J PROD ECON* 77 (June 2002), pp. 299–313. doi: 10.1016/S0925-5273(01)00156-6.
- [36] Gamax Laboratory Solutions. *Maintenance Strategies*. <https://gamaxlabsol.com/matlab/matlab-for-industry/predictive-maintenance/>.
- [37] R Keith Mobley. *An introduction to predictive maintenance*. Elsevier, 2002.

- [38] Jamie Coble. "Merging Data Sources to Predict Remaining Useful Life—An Automated Method to Identify Prognostic Parameters". In: (Aug. 2022).
- [39] Rodrigo Barbosa de Santis and Marcelo Azevedo Costa. "Extended Isolation Forests for Fault Detection in Small Hydroelectric Plants". In: *Sustainability* 12.16 (2020). ISSN: 2071-1050. DOI: 10.3390/su12166421.
- [40] Alessandro Betti, Emanuele Crisostomi, Gianluca Paolinelli, Antonio Piazzzi, Fabrizio Ruffini, and Mauro Tucci. "Condition monitoring and predictive maintenance methodologies for hydropower plants equipment". In: *Renewable Energy* 171 (2021), pp. 246–253.
- [41] Nishchal K Verma, Vishal Kumar Gupta, Mayank Sharma, and Rahul Kumar Sevakula. "Intelligent condition based monitoring of rotating machines using sparse auto-encoders". In: *2013 IEEE Conference on prognostics and health management (PHM)*. IEEE. 2013, pp. 1–7.
- [42] Thyago P Carvalho, Fabrizzio AAMN Soares, Roberto Vita, Roberto da P Francisco, João P Basto, and Symone GS Alcalá. "A systematic literature review of machine learning methods applied to predictive maintenance". In: *Computers & Industrial Engineering* 137 (2019), p. 106024.
- [43] Gian Antonio Susto, Andrea Schirru, Simone Pampuri, Seán McLoone, and Alessandro Beghi. "Machine Learning for Predictive Maintenance: A Multiple Classifier Approach". In: *IEEE Transactions on Industrial Informatics* 11.3 (2015), pp. 812–820. DOI: 10.1109/TII.2014.2349359.
- [44] mrbullwinkle, nitinme, jr-MS, YutongTie-MSFT, and yingqunpku. *What is Multivariate Anomaly Detector? (Public Preview)*. <https://docs.microsoft.com/en-us/azure/cognitive-services/anomaly-detector/overview-multivariate>. (Visited on 08/08/2025).
- [45] K Vereide, U Stevanovic, E Digerud, C Blancarte, and R Perez. "Strategy and application of new technologies for condition monitoring and predictive maintenance of hydro units". In: ().
- [46] Farzaneh Ahmadzadeh and Jan Lundberg. "Remaining useful life estimation". In: *International Journal of System Assurance Engineering and Management* 5.4 (2014), pp. 461–474.

BIBLIOGRAPHY

- [47] Bernhard P. Bettig and Ray P. S. Han. "Predictive Maintenance Using the Rotordynamic Model of a Hydraulic Turbine-Generator Rotor". In: *Journal of Vibration and Acoustics* 120.2 (Apr. 1998), pp. 441–448. ISSN: 1048-9002. DOI: 10.1115/1.2893849. eprint:https://asmedigitalcollection.asme.org/vibrationacoustics/article-pdf/120/2/441/5592537/441_1.pdf.
- [48] Runqing Huang, Lifeng Xi, Xinglin Li, C Richard Liu, Hai Qiu, and Jay Lee. "Residual life predictions for ball bearings based on self-organizing map and back propagation neural network methods". In: *Mechanical systems and signal processing* 21.1 (2007), pp. 193–207.
- [49] Wilson Q Wang, M Farid Golnaraghi, and Fathy Ismail. "Prognosis of machine health condition using neuro-fuzzy systems". In: *Mechanical Systems and Signal Processing* 18.4 (2004), pp. 813–831.
- [50] Mingyuan Zhang, Chen He, Chengxuan Huang, and Jianhong Yang. "A weighted time embedding transformer network for remaining useful life prediction of rolling bearing". In: *Reliability Engineering & System Safety* 251 (2024), p. 110399.
- [51] Dawn An, Joo-Ho Choi, and Nam Ho Kim. "Prediction of remaining useful life under different conditions using accelerated life testing data". In: *Journal of Mechanical Science and Technology* 32.6 (2018), pp. 2497–2507.
- [52] Cornelius Scheffer and Paresh Girdhar. *Practical machinery vibration analysis and predictive maintenance*. Elsevier, 2004.
- [53] Ernest C Fitch. *Proactive maintenance for mechanical systems*. 5. Elsevier, 2013.
- [54] John Moubray. *Reliability-centered maintenance*. Industrial Press Inc., 2001.
- [55] John S Gulliver and Roger EA Arndt. *Hydropower engineering handbook*. McGraw-Hill, Inc., 1991.
- [56] S Graham Kelly. *Advanced vibration analysis*. CRC Press, 2006.
- [57] Jeremy Wright. *What Causes Machinery Vibration?* <https://www.machinerylubrication.com/Read/25974/signs-tips-machinery-vibration>. (Visited on 08/08/2025).
- [58] ReliabilityWeb.COM. *The New Generation of Vibration Transmitters*. https://reliabilityweb.com/articles/entry/the_new_generation_of_vibration_transmitters. (Visited on 08/08/2025).

- [59] M Attia Hili, T Fakhfakh, and M Haddar. "Failure analysis of a misaligned and unbalanced flexible rotor". In: *Journal of Failure Analysis and Prevention* 6.4 (2006), pp. 73–82.
- [60] Molka Attia Hili, Tahar Fakhfakh, Lotfi Hammami, and Mohamed Haddar. "Shaft misalignment effect on bearings dynamical behavior". In: *The International Journal of Advanced Manufacturing Technology* 26.5 (2005), pp. 615–622.
- [61] Dragica Jošt and Andrej Lipej. "Numerical prediction of non-cavitating and cavitating vortex rope in a Francis turbine draft tube". In: *Strojniški vestnik-Journal of Mechanical Engineering* 57.6 (2011), pp. 445–456.
- [62] A Baya, S Muntean, V C Câmpian, A Cuzmoș, M Diaconescu, and G Bălan. "Experimental investigations of the unsteady flow in a Francis turbine draft tube cone". In: *IOP Conference Series: Earth and Environmental Science* 12 (Aug. 2010), p. 012007. doi: 10.1088/1755-1315/12/1/012007.
- [63] Andritz Hydro. *Property of Andritz Hydro, which granted us permission to use*. <https://www.andritz.com/hydro-en>. (Visited on 08/08/2025).
- [64] Alan M Turing and J Haugeland. "Computing machinery and intelligence". In: *The Turing Test: Verbal Behavior as the Hallmark of Intelligence* (1950), pp. 29–56.
- [65] Tom M Mitchell and Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997.
- [66] Nils J. Nilsson. *INTRODUCTION TO MACHINE LEARNING*. <https://ai.stanford.edu/~nilsson/MLBOOK.pdf>(visited 27-07-2022). 1998.
- [67] Leo Egghe. "Untangling Herdan's law and Heaps' law: Mathematical and informetric arguments". In: *Journal of the American Society for Information Science and Technology* 58.5 (2007), pp. 702–709.
- [68] David Ruppert. *The elements of statistical learning: data mining, inference, and prediction*. 2004.
- [69] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.
- [70] Grigorios Tsoumakas and Ioannis Katakis. "Multi-label classification: An overview". In: *Data Warehousing and Mining: Concepts, Methodologies, Tools, and Applications* (2008), pp. 64–74.
- [71] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist> 2 (2010).

BIBLIOGRAPHY

- [72] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. "O'Reilly Media, Inc.", 2022.
- [73] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.
- [74] Kai-Ying Chen, Long-Sheng Chen, Mu-Chen Chen, and Chia-Lung Lee. "Using SVM based method for equipment fault detection in a thermal power plant". In: *Computers in industry* 62.1 (2011), pp. 42–50.
- [75] Tzu-Liang Tseng, Kalyan Reddy Aleti, Zhonghua Hu, and Yongjin Kwon. "E-quality control: A support vector machines approach". In: *Journal of Computational Design and Engineering* 3.2 (2016), pp. 91–101.
- [76] Sotiris B Kotsiantis, Ioannis Zaharakis, P Pintelas, et al. "Supervised machine learning: A review of classification techniques". In: *Emerging artificial intelligence applications in computer engineering* 160.1 (2007), pp. 3–24.
- [77] Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27.
- [78] Haiya Wang, Zhongqing Yu, and Lu Guo. "Real-time online fault diagnosis of rolling bearings based on KNN algorithm". In: *Journal of Physics: Conference Series*. Vol. 1486. 3. IOP Publishing. 2020, p. 032019.
- [79] Wenbo Zhu, Wei Sun, and Jose Romagnoli. "Adaptive k-nearest-neighbor method for process monitoring". In: *Industrial & Engineering Chemistry Research* 57.7 (2018), pp. 2574–2586.
- [80] Fan Li and Guang Jin. "Research on power energy load forecasting method based on KNN". In: *International Journal of Ambient Energy* 43.1 (2022), pp. 946–951.
- [81] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.
- [82] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, et al. "Xgboost: extreme gradient boosting". In: *R package version 0.4-2* 1.4 (2015), pp. 1–4.
- [83] Inala Vivek Vamsi, Nippani Abhinav, Amar Kumar Verma, and Sudha Radhika. "Random forest based real time fault monitoring system for industries". In: *2018 4th International Conference on Computing Communication and Automation (ICCCA)*. IEEE. 2018, pp. 1–6.

- [84] Tianteng Wang, Xuping Wang, Ruize Ma, Xiaoyu Li, Xiangpei Hu, Felix TS Chan, and Junhu Ruan. "Random forest-bayesian optimization for product quality prediction with large-scale dimensions in process industrial cyber-physical systems". In: *IEEE Internet of Things Journal* 7.9 (2020), pp. 8641–8653.
- [85] Gérard Biau and Erwan Scornet. "A random forest guided tour". In: *Test* 25.2 (2016), pp. 197–227.
- [86] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. "A Survey of Methods for Explaining Black Box Models". In: *ACM Comput. Surv.* 51.5 (Aug. 2018). issn: 0360-0300. doi: 10.1145/3236009.
- [87] Charu C Aggarwal and Chandan K Reddy. "Data clustering". In: *Algorithms and applications. Chapman&Hall/CRC Data mining and Knowledge Discovery series, Londra* (2014).
- [88] Ka-Chun Wong. "A Short Survey on Data Clustering Algorithms". In: *CoRR abs/1511.09123* (2015). arXiv: 1511.09123. URL: <http://arxiv.org/abs/1511.09123>.
- [89] Stuart Lloyd. "Least squares quantization in PCM". In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.
- [90] D. Comaniciu and P. Meer. "Mean shift: a robust approach toward feature space analysis". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.5 (2002), pp. 603–619. doi: 10.1109/34.1000236.
- [91] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.
- [92] Iva Matetić, Ivan Štajduhar, Igor Wolf, and Sandi Ljubic. "A Review of Data-Driven Approaches and Techniques for Fault Detection and Diagnosis in HVAC Systems". In: *Sensors* 23.1 (2023). issn: 1424-8220. doi: 10.3390/s23010001.
- [93] Christos T Yiakopoulos, Konstantinos C Gryllias, and Ioannis A Antoniadis. "Rolling element bearing fault detection in industrial environments based on a K-means clustering approach". In: *Expert Systems with Applications* 38.3 (2011), pp. 2888–2911.
- [94] Laurens Van Der Maaten, Eric O Postma, H Jaap Van Den Herik, et al. "Dimensionality reduction: A comparative review". In: *Journal of machine learning research* 10.66-71 (2009), p. 13.

BIBLIOGRAPHY

- [95] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2.11 (1901), pp. 559–572.
- [96] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.
- [97] Leland McInnes, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction". In: *arXiv preprint arXiv:1802.03426* (2018).
- [98] Mark A Kramer. "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE journal* 37.2 (1991), pp. 233–243.
- [99] Liang Zhang and Lingyun Liu. "Data anomaly detection based on isolation forest algorithm". In: *2022 International Conference on Computation, Big-Data and Engineering (ICCBDE)*. IEEE. 2022, pp. 87–89.
- [100] Prateek Chhikara, Nikhil Jain, Rajkumar Tekchandani, and Neeraj Kumar. "Data dimensionality reduction techniques for Industry 4.0: Research results, challenges, and future research directions". In: *Software: Practice and Experience* 52.3 (2022), pp. 658–688. DOI: <https://doi.org/10.1002/spe.2876>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2876>.
- [101] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.
- [102] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the royal statistical society: series B (methodological)* 39.1 (1977), pp. 1–22.
- [103] Emanuel Parzen. "On estimation of a probability density function and mode". In: *The annals of mathematical statistics* 33.3 (1962), pp. 1065–1076.
- [104] Longin Jan Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. "Outlier detection with kernel density functions". In: *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer. 2007, pp. 61–75.
- [105] Muhammad Syafiq Mohd Pozi and Mohd. Hasbullah Omar. "A Kernel Density Estimation Method to Generate Synthetic Shifted Datasets in Privacy-Preserving Task." In: *J. Internet Serv. Inf. Secur.* 10.4 (2020), pp. 70–89.
- [106] Johannes Fürnkranz and Tomáš Kliegr. "A brief overview of rule learning". In: *International symposium on rules and rule markup languages for the semantic web*. Springer. 2015, pp. 54–69.

- [107] P.D. McNicholas, T.B. Murphy, and M. O'Regan. "Standardising the lift of an association rule". In: *Computational Statistics & Data Analysis* 52.10 (2008), pp. 4712–4721. ISSN: 0167-9473. DOI: <https://doi.org/10.1016/j.csda.2008.03.013>.
- [108] Rakesh Agrawal and Ramakrishnan Srikant. "Fast algorithms forming association rules". In: *20th VLDB Conference*. 1994, pp. 487–499.
- [109] Manpreet Kaur and Shivani Kang. "Market Basket Analysis: Identify the changing trends of market data using association rule mining". In: *Procedia computer science* 85 (2016), pp. 78–85.
- [110] Tyler McCormick, Cynthia Rudin, and David Madigan. "A hierarchical model for association rule mining of sequential events: An approach to automated medical symptom prediction". In: (2011).
- [111] Flora S Tsai. "Network intrusion detection using association rules". In: *International journal of recent trends in engineering* 2.2 (2009), p. 202.
- [112] Jesper E Van Engelen and Holger H Hoos. "A survey on semi-supervised learning". In: *Machine learning* 109.2 (2020), pp. 373–440.
- [113] Xiaojin Jerry Zhu. "Semi-supervised learning literature survey". In: (2005).
- [114] Herbert Robbins. "Some aspects of the sequential design of experiments". In: (1952).
- [115] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1 (1988), pp. 9–44.
- [116] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3 (1992), pp. 279–292.
- [117] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.
- [118] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).
- [119] Alaa Eatzaz Abdalla. "A reinforcement learning algorithm for operations planning of a hydroelectric power multireservoir system". PhD thesis. University of British Columbia, 2007.

BIBLIOGRAPHY

- [120] Tadanobu Inoue, Giovanni De Magistris, Asim Munawar, Tsuyoshi Yokoya, and Ryuki Tachibana. “Deep reinforcement learning for high precision assembly tasks”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 819–825.
- [121] Rajesh Siraskar, Satish Kumar, Shruti Patil, Arunkumar Bongale, and Ketan Kotecha. “Reinforcement learning for predictive maintenance: A systematic technical review”. In: *Artificial Intelligence Review* 56.11 (2023), pp. 12885–12947.
- [122] Oguzhan Dogru, Junyao Xie, Om Prakash, Ranjith Chiplunkar, Jansen Soesanto, Hongtian Chen, Kirubakaran Velswamy, Fadi Ibrahim, and Biao Huang. “Reinforcement Learning in Process Industries: Review and Perspective”. In: *IEEE/CAA Journal of Automatica Sinica* 11.2 (2024), pp. 283–300. doi: 10.1109/JAS.2024.124227.
- [123] Benjamin Rolf, Ilya Jackson, Marcel Müller, Sebastian Lang, Tobias Reggelin, and Dmitry Ivanov. “A review on reinforcement learning algorithms and applications in supply chain management”. In: *International Journal of Production Research* 61.20 (2023), pp. 7151–7179.
- [124] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.
- [125] Jiahao Shi, Zhenye Li, Tingting Zhu, Dongyi Wang, and Chao Ni. “Defect detection of industry wood veneer based on NAS and multi-channel mask R-CNN”. In: *Sensors* 20.16 (2020), p. 4398.
- [126] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
- [127] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).
- [128] Chanthol Eang and Seungjae Lee. “Predictive maintenance and fault detection for motor drive control systems in industrial robots using CNN-RNN-based observers”. In: *Sensors* 25.1 (2024), p. 25.
- [129] Ibtissam Amalou, Naoual Mouhni, and Abdelmounaim Abdali. “Multivariate time series prediction by RNN architectures for energy consumption forecasting”. In: *Energy Reports* 8 (2022), pp. 1084–1091.

- [130] Shuijing Liu, Peixin Chang, Weihang Liang, Neeloy Chakraborty, and Katherine Driggs-Campbell. “Decentralized structural-rnn for robot crowd navigation with deep reinforcement learning”. In: *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2021, pp. 3517–3524.
- [131] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.
- [132] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.
- [133] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly detection: A survey”. In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.
- [134] JWL Glaisher. “On the rejection of discordant observations”. In: *Monthly Notices of the Royal Astronomical Society* 33 (1873), pp. 391–402.
- [135] Francis Ysidro Edgeworth. “Xli. on discordant observations”. In: *The london, edinburgh, and dublin philosophical magazine and journal of science* 23.143 (1887), pp. 364–375.
- [136] Gian Antonio Susto, Matteo Terzi, Chiara Masiero, Simone Pampuri, and Andrea Schirru. “A Fraud Detection Decision Support System via Human On-Line Behavior Characterization and Machine Learning”. In: *2018 First International Conference on Artificial Intelligence for Industries (AI4I)*. 2018, pp. 9–14. doi: 10.1109/AI4I.2018.8665694.
- [137] Yousof Al-Hammadi, Uwe Aickelin, and Julie Greensmith. “DCA for bot detection”. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. 2008, pp. 1807–1816. doi: 10.1109/CEC.2008.4631034.
- [138] Mattia Carletti, Chiara Masiero, Alessandro Beghi, and Gian Antonio Susto. “Explainable Machine Learning in Industry 4.0: Evaluating Feature Importance in Anomaly Detection to Enable Root Cause Analysis”. In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. 2019, pp. 21–26. doi: 10.1109/SMC.2019.8913901.
- [139] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. “Ganomaly: Semi-supervised anomaly detection via adversarial training”. In: *Asian conference on computer vision*. Springer. 2018, pp. 622–637.
- [140] Chun-Liang Li, Kihyuk Sohn, Jinsung Yoon, and Tomas Pfister. “Cutpaste: Self-supervised learning for anomaly detection and localization”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 9664–9674.

BIBLIOGRAPHY

- [141] MJ Desforges, PJ Jacob, and JE Cooper. “Applications of probability density estimation to the detection of abnormal conditions in engineering”. In: *Proceedings of the institution of mechanical engineers, part c: Journal of mechanical engineering science* 212.8 (1998), pp. 687–703.
- [142] Zheng Li, Yue Zhao, Xiyang Hu, Nicola Botta, Cezar Ionescu, and George Chen. “Ecod: Unsupervised outlier detection using empirical cumulative distribution functions”. In: *IEEE Transactions on Knowledge and Data Engineering* (2022).
- [143] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. “LOF: identifying density-based local outliers”. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 93–104.
- [144] Yahya Almardeny, Noureddine Boujnah, and Frances Cleary. “A novel outlier detection method for multivariate data”. In: *IEEE Transactions on Knowledge and Data Engineering* (2020).
- [145] David MJ Tax and Robert PW Duin. “Support vector data description”. In: *Machine learning* 54.1 (2004), pp. 45–66.
- [146] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation forest”. In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, pp. 413–422.
- [147] Yue Zhao and Maciej K Hryniewicki. “XGBOD: improving supervised outlier detection with unsupervised representation learning”. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, pp. 1–8.
- [148] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. *Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection*. 2018. DOI: 10.48550/ARXIV.1802.09089.
- [149] Pooja Kamat and Rekha Sugandhi. “Anomaly detection for predictive maintenance in industry 4.0-A survey”. In: *E3S web of conferences*. Vol. 170. EDP Sciences. 2020, p. 02007.
- [150] Jiaqi Lyu and Souran Manoochehri. “Online convolutional neural network-based anomaly detection and quality control for fused filament fabrication process”. In: *Virtual and Physical Prototyping* 16.2 (2021), pp. 160–177.
- [151] Francesca Calabrese, Alberto Regattieri, Marco Bortolini, and Francesco Gabriele Galizia. “Data-driven fault detection and diagnosis: Challenges and opportunities in real-world scenarios”. In: *Applied Sciences* 12.18 (2022), p. 9212.

- [152] Chris M Bishop. “Training with noise is equivalent to Tikhonov regularization”. In: *Neural computation* 7.1 (1995), pp. 108–116.
- [153] Huyen Chip. *Designing machine learning systems: An iterative process for production-ready applications*. 2022.
- [154] Burr Settles. “Active learning literature survey”. In: (2009).
- [155] Steven CH Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. “Online learning: A comprehensive survey”. In: *Neurocomputing* 459 (2021), pp. 249–289.
- [156] Dominic Gorecky, Mathias Schmitt, Matthias Loskyll, and Detlef Zühlke. “Human-machine-interaction in the industry 4.0 era”. In: *2014 12th IEEE international conference on industrial informatics (INDIN)*. Ieee. 2014, pp. 289–294.
- [157] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. 2022. URL: <https://christophm.github.io/interpretable-ml-book>.
- [158] Eduardo Mosqueira-Rey, Elena Hernández-Pereira, David Alonso-Ríos, José Bobes-Bascarán, and Ángel Fernández-Leal. “Human-in-the-loop machine learning: a state of the art”. In: *Artificial Intelligence Review* 56.4 (2023), pp. 3005–3054.
- [159] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. “Power to the people: The role of humans in interactive machine learning”. In: *AI magazine* 35.4 (2014), pp. 105–120.
- [160] Gonzalo Ramos, Christopher Meek, Patrice Simard, Jina Suh, and Soroush Ghorashi. “Interactive machine teaching: a human-centered approach to building machine-learned models”. In: *Human–Computer Interaction* 35.5-6 (2020), pp. 413–451.
- [161] Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. “Content-Based Image Retrieval at the End of the Early Years”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.12 (2000), pp. 1349–1380. DOI: 10.1109/34.895972.
- [162] Jignesh Shah, Jing Hu, Oreane Y Edelenbosch, and Michelle TH van Vliet. “Global dataset combining open-source hydropower plant and reservoir data”. In: *Scientific data* 12.1 (2025), p. 646.
- [163] Yasir Saleem Afridi. “Bearing Vibration Dataset of a Hydropower Project”. In: (Oct. 2022). DOI: 10.6084/m9.figshare.21290895.v1.

BIBLIOGRAPHY

- [164] Fabio Martinelli, Francesco Mercaldo, Albina Orlando, Vittoria Nardone, Antonella Santone, and Arun Kumar Sangaiah. "Human behavior characterization for driving style recognition in vehicle system". In: *Computers & Electrical Engineering* 83 (2020), p. 102504.
- [165] Shebuti Rayana. *ODDS Library*. 2016. URL: <https://shebuti.com/outlier-detection-datasets-odds/> (visited on 08/08/2025).
- [166] UCI Machine Learning Repository. *Statlog (Landsat Satellite) Data Set*. <https://archive.ics.uci.edu/dataset/146/statlog+landsat+satellite>. 2025. (Visited on 08/08/2025).
- [167] UCI Machine Learning Repository. *Glass Identification Data Set*. <https://archive.ics.uci.edu/dataset/42/glass+identification>. 2025. (Visited on 08/08/2025).
- [168] Elisa Marcelli, Tommaso Barbariol, Davide Sartor, and Gian Antonio Susto. "Active Learning-based Isolation Forest (ALIF): Enhancing anomaly detection with expert feedback". In: *Information Sciences* 678 (2024), p. 121012. ISSN: 0020-0255. DOI: <https://doi.org/10.1016/j.ins.2024.121012>.
- [169] Davide Sartor, Tommaso Barbariol, and Gian Antonio Susto. "Bayesian active learning isolation forest (B-ALIF): A weakly supervised strategy for anomaly detection". In: *Engineering Applications of Artificial Intelligence* 130 (2024), p. 107671. ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2023.107671>.
- [170] Pierre Baldi. "Autoencoders, unsupervised learning, and deep architectures". In: *Proceedings of ICML workshop on unsupervised and transfer learning*. JMLR Workshop and Conference Proceedings. 2012, pp. 37–49.
- [171] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. DOI: [10.48550/ARXIV.1312.6114](https://arxiv.org/abs/1312.6114).
- [172] Elad Plaut. *From Principal Subspaces to Principal Components with Linear Autoencoders*. 2018. DOI: [10.48550/ARXIV.1804.10253](https://arxiv.org/abs/1804.10253).
- [173] Saket Sathe and Charu C. Aggarwal. "Subspace Outlier Detection in Linear Time with Randomized Hashing". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 459–468. DOI: [10.1109/ICDM.2016.0057](https://doi.org/10.1109/ICDM.2016.0057).
- [174] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. "Deep One-Class Classification". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 4393–4402.

- [175] Sam Corbett-Davies and Sharad Goel. "The measure and mismeasure of fairness: A critical review of fair machine learning". In: *arXiv preprint arXiv:1808.00023* (2018).
- [176] Finale Doshi-Velez and Been Kim. "Towards a rigorous science of interpretable machine learning". In: *arXiv preprint arXiv:1702.08608* (2017).
- [177] Mattia Carletti, Matteo Terzi, and Gian Antonio Susto. "Interpretable anomaly detection with diffi: Depth-based isolation forest feature importance". In: *arXiv preprint arXiv:2007.11117* (2020).
- [178] Qibo Yang, Jaskaran Singh, and Jay Lee. "Isolation-based feature selection for unsupervised outlier detection". In: *Proc. Annu. Conf. Progn. Health Manag. Soc.* Vol. 11. 2019.
- [179] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "'Why should i trust you?' Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.
- [180] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.
- [181] Lloyd S Shapley, Alvin E Roth, et al. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.
- [182] Sungsoo Ray Hong, Jessica Hullman, and Enrico Bertini. "Human factors in model interpretability: Industry practices, challenges, and needs". In: *Proceedings of the ACM on Human-Computer Interaction* 4.CSCW1 (2020), pp. 1–26.
- [183] slundberg. *boston_waterfall.png*. <https://github.com/slundberg/shap>. (Visited on 08/08/2025).
- [184] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. 2016. DOI: 10.48550/ARXIV.1602.04938.
- [185] Rodrigo Barbosa de Santis and Marcelo Azevedo Costa. "Extended isolation forests for fault detection in small hydroelectric plants". In: *Sustainability* 12.16 (2020), p. 6421.
- [186] Wenlong Zhu, Jianzhong Zhou, Xin Xia, Chaoshun Li, Jian Xiao, Han Xiao, and Xinxin Zhang. "A novel KICA-PCA fault detection model for condition process of hydroelectric generating unit". In: *Measurement* 58 (2014), pp. 197–206. ISSN: 0263-2241. DOI: <https://doi.org/10.1016/j.measurement.2014.08.026>.

BIBLIOGRAPHY

- [187] Sahand Hariri, Matias Carrasco Kind, and Robert J Brunner. “Extended isolation forest”. In: *IEEE transactions on knowledge and data engineering* 33.4 (2019), pp. 1479–1489.
- [188] Hongteng Wang, Xuwei Liu, Liyong Ma, and Yong Zhang. “Anomaly detection for hydropower turbine unit based on variational modal decomposition and deep autoencoder”. In: *Energy Reports* 7 (2021), pp. 938–946.
- [189] Mattia Carletti, Matteo Terzi, and Gian Antonio Susto. “Interpretable anomaly detection with diffi: Depth-based feature importance of isolation forest”. In: *Engineering Applications of Artificial Intelligence* 119 (2023), p. 105730.
- [190] Suk Hoon Choi and Jong Min Lee. “Explainable fault diagnosis model using stacked autoencoder and kernel SHAP”. In: *2022 IEEE International Symposium on Advanced Control of Industrial Processes (AdCONIP)*. IEEE. 2022, pp. 182–187.
- [191] Lucas C Brito, Gian Antonio Susto, Jorge N Brito, and Marcus AV Duarte. “An explainable artificial intelligence approach for unsupervised fault detection and diagnosis in rotating machinery”. In: *Mechanical Systems and Signal Processing* 163 (2022), p. 108105.
- [192] Thomas Michael Welte, Jørn Foros, MH Nielsen, and M Adsten. “MonitorX–Experience from a Norwegian-Swedish research project on industry 4.0 and digitalization applied to fault detection and maintenance of hydropower plants”. In: *Hydro 2018-Progress through partnership, Gdansk, Poland, 15-17 October 2018* (2018).
- [193] Christophe NICOLET, Nicolas Ruchonnet, and François Avellan. “One-dimensional modeling of rotor stator interaction in Francis pump-turbine”. In: *Proceedings of the 23rd IAHR Symposium on Hydraulic Machinery and Systems, Yokohama, Japan*. Vol. 1. CONF. International Association For Hydraulic Research. 2006, pp. 1–15.
- [194] *Python*. <https://www.python.org/>. (Visited on 08/08/2025).
- [195] *Python Outlier Detection package*. <https://pyod.readthedocs.io/en/latest/index.html>. (Visited on 08/08/2025).
- [196] *Python Streaming Anomaly Detection package*. <https://pysad.readthedocs.io/en/latest/index.html>. (Visited on 08/08/2025).
- [197] Diederik P Kingma. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [198] Flavio D Garcia, David F Oswald, Timo Kasper, and Pierre Pavlidès. “Lock It and Still Lose It-on the (In) Security of Automotive Remote Keyless Entry Systems.” In: *USENIX security symposium*. Vol. 53. 2016.

- [199] Aurélien Francillon, Boris Danev, and Srdjan Capkun. "Relay attacks on passive keyless entry and start systems in modern cars". In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Eidgenössische Technische Hochschule Zürich, Department of Computer Science. 2011.
- [200] Yong Goo Kang, Kyung Ho Park, and Huy Kang Kim. "Automobile theft detection by clustering owner driver data". In: *arXiv preprint arXiv:1909.08929* (2019).
- [201] Chandrasekar Ravi, Anmol Tigga, G Thippa Reddy, Saqib Hakak, and Mamoun Alazab. "Driver identification using optimized deep learning model in smart transportation". In: *ACM Transactions on Internet Technology* 22.4 (2022), pp. 1–17.
- [202] Pei-Yu Tseng, Po-Ching Lin, and Edy Kristianto. "Vehicle theft detection by generative adversarial networks on driving behavior". In: *Engineering Applications of Artificial Intelligence* 117 (2023), p. 105571. ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2022.105571>.
- [203] Xin Lin, Kai Zhang, Wangjing Cao, and Lin Zhang. "Driver Evaluation and Identification Based on Driving Behavior Data". In: *2018 5th International Conference on Information Science and Control Engineering (ICISCE)*. IEEE. 2018, pp. 718–722.
- [204] Abenezer Girma, Xuyang Yan, and Abdollah Homaifar. "Driver identification based on vehicle telematics data using lstm-recurrent neural network". In: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2019, pp. 894–902.
- [205] Yijie Xun, Jiajia Liu, Nei Kato, Yongqiang Fang, and Yanning Zhang. "Automobile driver fingerprinting: A new machine learning based authentication scheme". In: *IEEE Transactions on Industrial Informatics* 16.2 (2019), pp. 1417–1426.
- [206] Abdellah El Mekki, Afaf Bouhoute, and Ismail Berrada. "Improving driver identification for the next-generation of in-vehicle software systems". In: *IEEE Transactions on Vehicular Technology* 68.8 (2019), pp. 7406–7415.
- [207] Bernhard Gahr, Shu Liu, Kevin Koch, Filipe Barata, André Dahlinger, Benjamin Ryder, Elgar Fleisch, and Felix Wortmann. "Driver identification via the steering wheel". In: *arXiv preprint arXiv:1909.03953* (2019).
- [208] Mozghan Nasr Azadani and Azzedine Boukerche. "Performance evaluation of driving behavior identification models through can-bus data". In: *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2020, pp. 1–6.

BIBLIOGRAPHY

- [209] Hashim Abu-Gellban, Long Nguyen, Mahdi Moghadasi, Zhenhe Pan, and Fang Jin. “Livedi: An anti-theft model based on driving behavior”. In: *Proceedings of the 2020 ACM Workshop on Information Hiding and Multimedia Security*. 2020, pp. 67–72.
- [210] Emad Efatinasab, Francesco Marchiori, Denis Donadel, Alessandro Brighente, and Mauro Conti. “When Authentication Is Not Enough: On the Security of Behavioral-Based Driver Authentication Systems”. In: *arXiv preprint arXiv:2306.05923* (2024). arXiv: 2306.05923 [cs.CR].
- [211] Emad Efatinasab, Denis Donadel, and Mauro Conti. *GAN-CAN: A Novel Attack to Behavior-Based Driver Authentication Systems*. <https://hdl.handle.net/20.500.12608/52324>, In Master Thesis in Computer Science, Department of Mathematics “Tullio Levi-Civita”. University of Padova. 2022. (Visited on 08/08/2025).
- [212] Michael McCloskey and Neal J Cohen. “Catastrophic interference in connectionist networks: The sequential learning problem”. In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.
- [213] Zhizhong Li and Derek Hoiem. “Learning without forgetting”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.12 (2017), pp. 2935–2947.
- [214] Aslıhan Cura, Haluk Küçük, Erdem Ergen, and Ismail Burak Öksüzoğlu. “Driver Profiling Using Long Short Term Memory (LSTM) and Convolutional Neural Network (CNN) Methods”. In: *IEEE Transactions on Intelligent Transportation Systems* 22.10 (Oct. 2021). Conference Name: IEEE Transactions on Intelligent Transportation Systems, pp. 6572–6582. ISSN: 1558-0016. DOI: 10.1109/TITS.2020.2995722.
- [215] Kashmir Hill. *Automakers Are Sharing Consumers’ Driving Behavior With Insurance Companies*. www.nytimes.com/2024/03/11/technology/carmakers-driver-tracking-insurance.html. 2024.
- [216] Byung Il Kwak, JiYoung Woo, and Huy Kang Kim. “Know your master: Driver profiling-based anti-theft method”. In: *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. IEEE. 2016, pp. 211–218.
- [217] Mussadiq Abdul Rahim, Liehuang Zhu, Xin Li, Jiamou Liu, Zijian Zhang, Zhan Qin, Salabat Khan, and Keke Gai. “Zero-to-stable driver identification: A non-intrusive and scalable driver identification scheme”. In: *IEEE transactions on vehicular technology* 69.1 (2019), pp. 163–171.

- [218] Gabriela Ahmadi-Assalemi, Haider M Al-Khateeb, Carsten Maple, Gregory Epiphaniou, Mohammad Hammoudeh, Hamid Jahankhani, and Prashant Pillai. "Optimising driver profiling through behaviour modelling of in-car sensor and global positioning system data". In: *Computers & Electrical Engineering* 91 (2021), p. 107047.
- [219] Yijie Xun, Jiajia Liu, Nei Kato, Yongqiang Fang, and Yanning Zhang. "Automobile Driver Fingerprinting: A New Machine Learning Based Authentication Scheme". In: vol. 16. 2. 2020, pp. 1417–1426. doi: 10.1109/TII.2019.2946626.
- [220] Engin Erzin, Yücel Yemez, A Murat Tekalp, Aytül Erçil, Hakan Erdogan, and Hüseyin Abut. "Multimodal person recognition for human-vehicle interaction". In: *IEEE MultiMedia* 13.2 (2006), pp. 18–31.
- [221] David Hallac, Abhijit Sharang, Rainer Stahlmann, Andreas Lamprecht, Markus Huber, Martin Roehder, Jure Leskovec, et al. "Driver identification using automobile sensor data from a single turn". In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2016, pp. 953–958.
- [222] Saad Ezzini, Ismail Berrada, and Mounir Ghogho. "Who is behind the wheel? Driver identification and fingerprinting". In: *Journal of Big Data* 5.1 (2018), pp. 1–15.
- [223] Letizia Marchegiani and Ingmar Posner. "Long-term driving behaviour modelling for driver identification". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 913–919.
- [224] Angela Burton, Tapan Parikh, Shannon Mascarenhas, Jue Zhang, Jonathan Voris, N Sertac Artan, and Wenjia Li. "Driver identification and authentication with active behavior modeling". In: *2016 12th International Conference on Network and Service Management (CNSM)*. IEEE. 2016, pp. 388–393.
- [225] Mozhgan Nasr Azadani and Azzedine Boukerche. "Driver identification using vehicular sensing data: A deep learning approach". In: *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2021, pp. 1–6.
- [226] Jun Zhang, ZhongCheng Wu, Fang Li, Chengjun Xie, Tingting Ren, Jie Chen, and Liu Liu. "A deep learning framework for driving behavior identification on in-vehicle CAN-BUS sensor data". In: *Sensors* 19.6 (2019), p. 1356.
- [227] Jie Chen, ZhongCheng Wu, and Jun Zhang. "Driver identification based on hidden feature extraction by using adaptive nonnegativity-constrained autoencoder". In: *Applied Soft Computing* 74 (2019), pp. 1–9.

BIBLIOGRAPHY

- [228] Eric Coleman, Luigi Quarantiello, Ziyue Liu, Qinwen Yang, Samrat Mukherjee, Julio Hurtado, and Vincenzo Lomonaco. “Parameter-Efficient Continual Fine-Tuning: A Survey”. In: (Jan. 2025). DOI: 10.2139/ssrn.5510257.
- [229] Gido M. van de Ven and Andreas S. Tolias. “Three scenarios for continual learning”. In: *CoRR* abs/1904.07734 (2019). arXiv: 1904.07734. URL: <http://arxiv.org/abs/1904.07734>.
- [230] Vincenzo Lomonaco and Davide Maltoni. “Core50: a new dataset and benchmark for continuous object recognition”. In: *Conference on robot learning*. PMLR. 2017, pp. 17–26.
- [231] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. “Experience replay for continual learning”. In: *Advances in Neural Information Processing Systems* 32 (2019).
- [232] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.
- [233] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* (2015).
- [234] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. “Progressive neural networks”. In: *arXiv preprint arXiv:1606.04671* (2016).
- [235] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. “Pathnet: Evolution channels gradient descent in super neural networks”. In: *arXiv preprint arXiv:1701.08734* (2017).
- [236] Arun Mallya and Svetlana Lazebnik. “Packnet: Adding multiple tasks to a single network by iterative pruning”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 7765–7773.
- [237] Albin Soutif–Cormerais, Antonio Carta, Andrea Cossu, Julio Hurtado, Hamed Hemati, Vincenzo Lomonaco, and Joost Van de Weijer. *A Comprehensive Empirical Evaluation on Online Continual Learning*. 2023. arXiv: 2308.10328 [cs.LG]. URL: <https://arxiv.org/abs/2308.10328>.
- [238] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. *Dark Experience for General Continual Learning: a Strong, Simple Baseline*. 2020. arXiv: 2004.07211 [stat.ML]. URL: <https://arxiv.org/abs/2004.07211>.

- [239] S Hochreiter. “Long Short-term Memory”. In: *Neural Computation MIT-Press* (1997).
- [240] Infineon Technologies AG. *AURIX™ Family – TC39xXX*. <https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc3xx/aurix-family-tc39xxx/>. (Visited on 08/08/2025).
- [241] Kyle DeMedeiros, Abdeltawab Hendawi, and Marco Alvarez. “A survey of AI-based anomaly detection in IoT and sensor networks”. In: *Sensors* 23.3 (2023), p. 1352.
- [242] Waleed Hilal, S Andrew Gadsden, and John Yawney. “Financial fraud: a review of anomaly detection techniques and recent advances”. In: *Expert systems With applications* 193 (2022), p. 116429.
- [243] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. “Towards total recall in industrial anomaly detection”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 14318–14328.
- [244] Yue Zhao, Xiyang Hu, Cheng Cheng, Cong Wang, Changlin Wan, Wen Wang, Jianing Yang, Haoping Bai, Zheng Li, Cao Xiao, et al. “Suod: Accelerating large-scale unsupervised heterogeneous outlier detection”. In: *Proceedings of Machine Learning and Systems* 3 (2021), pp. 463–478.
- [245] Iker Antonio Olarra Maldonado, Erik Meeuwissen, Puck de Haan, and Rob van der Mei. “Telosian: Reducing False Positives in Real-Time Cyber Anomaly Detection by Fast Adaptation to Concept Drift.” In: *ICISSP (2)*. 2025, pp. 84–97.
- [246] Michael Heigl, Kumar Ashutosh Anand, Andreas Urmann, Dalibor Fiala, Martin Schramm, and Robert Hable. “On the improvement of the isolation forest algorithm for outlier detection with streaming data”. In: *Electronics* 10.13 (2021), p. 1534.
- [247] Ali Anaissi, Nguyen Lu Dang Khoa, Thierry Rakotoarivelo, Mehrisadat Makki Alamdari, and Yang Wang. “Adaptive online one-class support vector machines with applications in structural health monitoring”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 9.6 (2018), pp. 1–20.
- [248] Marius Dragoi, Elena Burceanu, Emanuela Haller, Andrei Manolache, and Florin Brad. “Anoshift: A distribution shift benchmark for unsupervised anomaly detection”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 32854–32867.
- [249] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. “Dynamic feature space and incremental feature selection for the classification of textual data streams”. In: *Knowledge Discovery from Data Streams* (2006), pp. 107–116.

BIBLIOGRAPHY

- [250] Xindong Wu, Kui Yu, Wei Ding, Hao Wang, and Xingquan Zhu. “Online feature selection with streaming features”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.5 (2012), pp. 1178–1192.
- [251] Emaad Manzoor, Hemank Lamba, and Leman Akoglu. “xstream: Outlier detection in feature-evolving data streams”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 1963–1972.
- [252] João Carvalho, Mengtao Zhang, Robin Geyer, Carlos Cotrini, and Joachim M Buhmann. “Invariant anomaly detection under distribution shifts: a causal perspective”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 56310–56337.
- [253] Danlei Li, Nirmal-Kumar C Nair, and Kevin I-Kai Wang. “ADDAEIL: Anomaly Detection with Drift-Aware Ensemble-Based Incremental Learning”. In: *Algorithms* 18.6 (2025), p. 359.
- [254] Feng Xiao and Jicong Fan. “Unsupervised anomaly detection in the presence of missing values”. In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 138130–138162.
- [255] Stephen Wambura, Jianbin Huang, and He Li. “Robust anomaly detection in feature-evolving time series”. In: *The Computer Journal* 65.5 (2022), pp. 1242–1256.
- [256] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- [257] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. *Explainable AI for Trees: From Local Explanations to Global Understanding*. 2019. arXiv: 1905.04610 [cs.LG]. URL: <https://arxiv.org/abs/1905.04610>.
- [258] Ferenc Móricz and Ulrich Stadtmüller. “Characterization of the convergence of weighted averages of sequences and functions”. In: *Periodica Mathematica Hungarica* 65.1 (2012), pp. 135–145.
- [259] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [260] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. “Array programming with NumPy”. In: *nature* 585.7825 (2020), pp. 357–362.

- [261] SHAP - SHapley Additive exPlanations package for Python. <https://shap.readthedocs.io/en/latest/>. (Visited on 08/08/2025).
- [262] Haoyu Wang, Chuanjiang Li, Peng Ding, Shaobo Li, Tandong Li, Chenyu Liu, Xiangjie Zhang, and Zejian Hong. "A novel transformer-based few-shot learning method for intelligent fault diagnosis with noisy labels under varying working conditions". In: *Reliability Engineering & System Safety* 251 (2024), p. 110400. ISSN: 0951-8320. DOI: <https://doi.org/10.1016/j.ress.2024.110400>.
- [263] H. He and E.A. Garcia. "Learning from Imbalanced Data". In: *IEEE Transactions on Knowledge and Data Engineering* 21 (2009), pp. 1263–1284.
- [264] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. "A systematic study of the class imbalance problem in convolutional neural networks". In: *Neural Networks* 106 (2018), pp. 249–259. ISSN: 0893-6080. DOI: <https://doi.org/10.1016/j.neunet.2018.07.011>.
- [265] Mohammed Khalilia, Sounak Chakraborty, and Mihail Popescu. "Predicting disease risks from highly imbalanced data using random forest". In: *BMC medical informatics and decision making* 11.1 (2011), p. 51.
- [266] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.
- [267] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. "ADASYN: Adaptive synthetic sampling approach for imbalanced learning". In: *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. Ieee. 2008, pp. 1322–1328.
- [268] Debashree Devi, Biswajit Purkayastha, et al. "Redundancy-driven modified Tomek-link based undersampling: A solution to class imbalance". In: *Pattern Recognition Letters* 93 (2017), pp. 3–12.
- [269] Dennis L Wilson. "Asymptotic properties of nearest neighbor rules using edited data". In: *IEEE Transactions on Systems, Man, and Cybernetics* 3 (2007), pp. 408–421.
- [270] Yao Ma, Hongbo Shi, Shuai Tan, Yang Tao, and Bing Song. "Consistency Regularization Auto-Encoder Network for Semi-Supervised Process Fault Diagnosis". In: *IEEE Transactions on Instrumentation and Measurement* 71.1 (2022), pp. 1–15. DOI: [10.1109/TIM.2022.3184346](https://doi.org/10.1109/TIM.2022.3184346).

BIBLIOGRAPHY

- [271] Manjiri Bhat and R.B. Keskar. "Self-supervised random forests for robust voice activity detection with limited labeled data". In: *Applied Acoustics* 234 (2025), p. 110636. ISSN: 0003-682X. DOI: <https://doi.org/10.1016/j.apacoust.2025.110636>.
- [272] Zhi-Wu Zhang, Xiao-Yuan Jing, and Tie-Jian Wang. "Label propagation based semi-supervised learning for software defect prediction". In: *Automated Software Engineering* 24.1 (2017), pp. 47–69.
- [273] Jingyao Wu, Zhibin Zhao, Chuang Sun, Ruqiang Yan, and Xuefeng Chen. "Few-shot transfer learning for intelligent fault diagnosis of machine". In: *Measurement* 166 (2020), p. 108202. ISSN: 0263-2241. DOI: <https://doi.org/10.1016/j.measurement.2020.108202>.
- [274] Shen Zhang, Fei Ye, Bingnan Wang, and Thomas G Habetler. "Few-shot bearing fault diagnosis based on model-agnostic meta-learning". In: *IEEE Transactions on Industry Applications* 57.5 (2021), pp. 4754–4764.
- [275] Cheng Cheng, Xiaoyu Liu, Beitong Zhou, and Ye Yuan. "Intelligent Fault Diagnosis With Noisy Labels via Semisupervised Learning on Industrial Time Series". In: *IEEE Transactions on Industrial Informatics* 19.6 (2023), pp. 7724–7732. DOI: [10.1109/TII.2022.3229130](https://doi.org/10.1109/TII.2022.3229130).
- [276] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. "MixMatch: A Holistic Approach to Semi-Supervised Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019.
- [277] Nantian Huang, Qingzhu Chen, Guowei Cai, Dianguo Xu, Liang Zhang, and Wenguang Zhao. "Fault Diagnosis of Bearing in Wind Turbine Gearbox Under Actual Operating Conditions Driven by Limited Data With Noise Labels". In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021), pp. 1–10. DOI: [10.1109/TIM.2020.3025396](https://doi.org/10.1109/TIM.2020.3025396).
- [278] Mohamed-Rafik Bouguelia, Slawomir Nowaczyk, KC Santosh, and Antanas Verikas. "Agreeing to disagree: active learning with noisy labels without crowdsourcing". In: *International journal of machine learning and cybernetics* 9.8 (2018), pp. 1307–1319.
- [279] Charles J Stone, Jerome Friedman, L Breiman, and R Olshen. "Classification and regression trees". In: *Wadsworth International Group* 8 (1984), pp. 452–456.
- [280] Ashwin Srinivasan. *Statlog (Landsat Satellite)*. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C55887>. 1993. URL: <https://doi.org/10.24432/C55887>.

- [281] B. German. *Glass Identification*. UCI Machine Learning Repository. DOI: <https://doi.org/10.2412/1987>. URL: <https://doi.org/10.24432/C5WW2P>.
- [282] Ziwei Chen, Yang Xu, Chao Zhang, and Min Tang. "Prediction of Glass Chemical Composition and Type Identification Based on Machine Learning Algorithms". In: *Applied Sciences* 14.10 (2024). ISSN: 2076-3417.
- [283] Rishabh Tiwari, Krishnateja Killamsetty, Rishabh Iyer, and Pradeep Shenoy. "Gcr: Gradient coreset based replay buffer selection for continual learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 99–108.
- [284] Gabriele Merlin, Vincenzo Lomonaco, Andrea Cossu, Antonio Carta, and Davide Bacciu. "Practical recommendations for replay-based continual learning methods". In: *International Conference on Image Analysis and Processing*. Springer. 2022, pp. 548–559.
- [285] Nikola Bugarin, Jovana Bugaric, Manuel Barusco, Davide Dalle Pezze, and Gian Antonio Susto. "Unveiling the Anomalies in an Ever-Changing World: A Benchmark for Pixel-Level Anomaly Detection in Continual Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 4065–4074.
- [286] David Lopez-Paz and Marc'Aurelio Ranzato. "Gradient episodic memory for continual learning". In: *Advances in neural information processing systems* 30 (2017).
- [287] Marina Ceccon, Davide Dalle Pezze, Alessandro Fabris, and Gian Antonio Susto. "Multi-Label Continual Learning for the Medical Domain: A Novel Benchmark". In: *arXiv preprint arXiv:2404.06859* (2024).



Additional Material for Continual Learning for Behavior-Based Driver Identification

A.1 HYPERPARAMETERS TUNING

This appendix provides detailed information about the tuning process of all the CL approaches, including plots of the hyperparameter search results, as well as our considerations and rationale behind the optimal configuration choices for each method. The results are based on a grid search using the values shown in the plots, following the training procedure reported in 5.4.2. Refer to Section 5.3 for theoretical background and relevant citations, and to Section 5.4.3 for details on the tuned hyperparameters.

A.1.1 ER

Given our resource-constrained environment, the most critical hyperparameter for this method is *Memory Size* (the size of the memory used for sampling rehearsal instances). Figure A.1a shows the accuracy for a *Memory Size* ranging from 100 to 4000. The performance improvement follows a logarithmic trend, as increased memory reduces forgetting until it reaches saturation. Values outside this range are impractical due to resource and performance constraints. Therefore, we select 1000 as the optimal memory size, balancing resource requirements and performance, as discussed in 5.4.3. This value is 10% of the training dataset, and aligns with common ranges found in the literature (1-10%) [283, 284, 285, 286, 287]. In Figure A.1b you can see that this parameter has no noticeable influence on the execution time.

A.1. HYPERPARAMETERS TUNING

The impact of the *Replay Ratio* is shown in Figure A.2a for accuracy and Figure A.2b for execution time. A ratio of 0.25 significantly reduces accuracy, while 0.75 offers slightly better accuracy than the canonical even mix but doubles the training time, as training instances processed per batch are halved. Therefore, we selected a ratio of 50% mix as the optimal parameter.

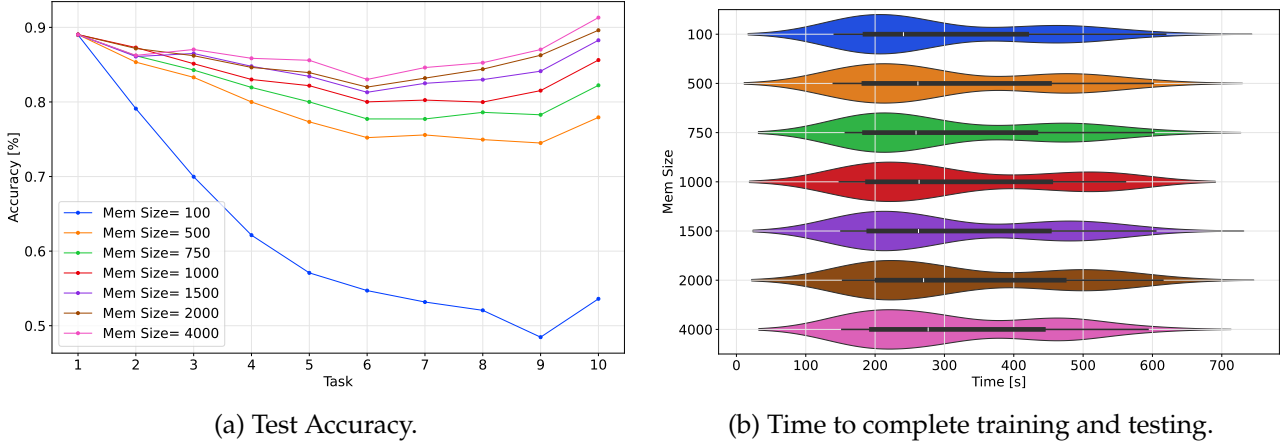


Figure A.1: Results of the *Memory Size* parameter search for the ER method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.

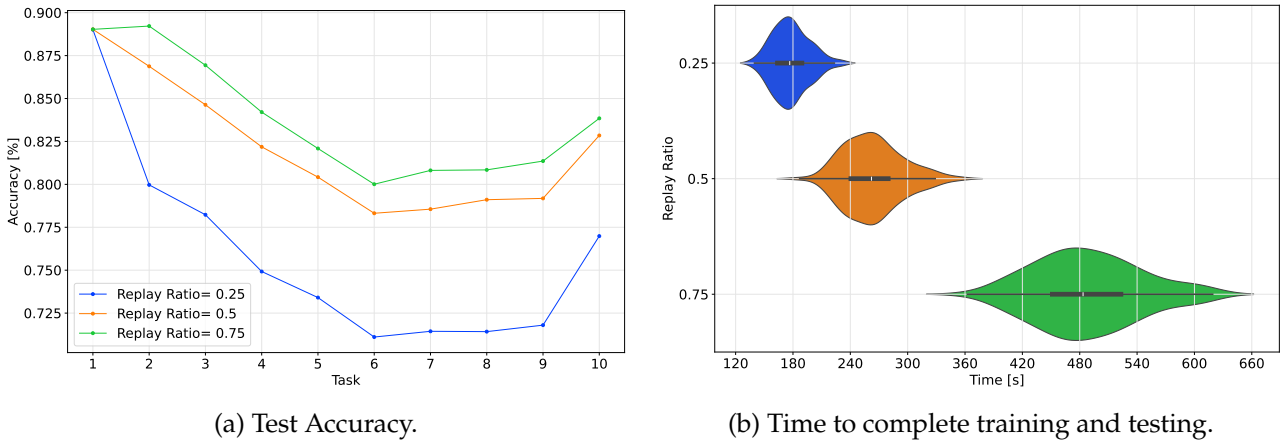


Figure A.2: Results of the *Replay Ratio* parameter search for the ER method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.

A.1.2 EWC

Figures A.3a and A.3b illustrate that varying the hyperparameter λ had minimal impact on both accuracy and execution time. In line with prior research, we explored the following range

of values: 1, 50, 100, 500, 1000, 5000, and 10000. Notably, the original study by Kirkpatrick et al. [232] identified 400 as the optimal value. After careful consideration, we selected $\lambda = 10000$, which struck an effective balance between the training and EWC losses, while also yielding a slight increase in accuracy compared to the other values tested.

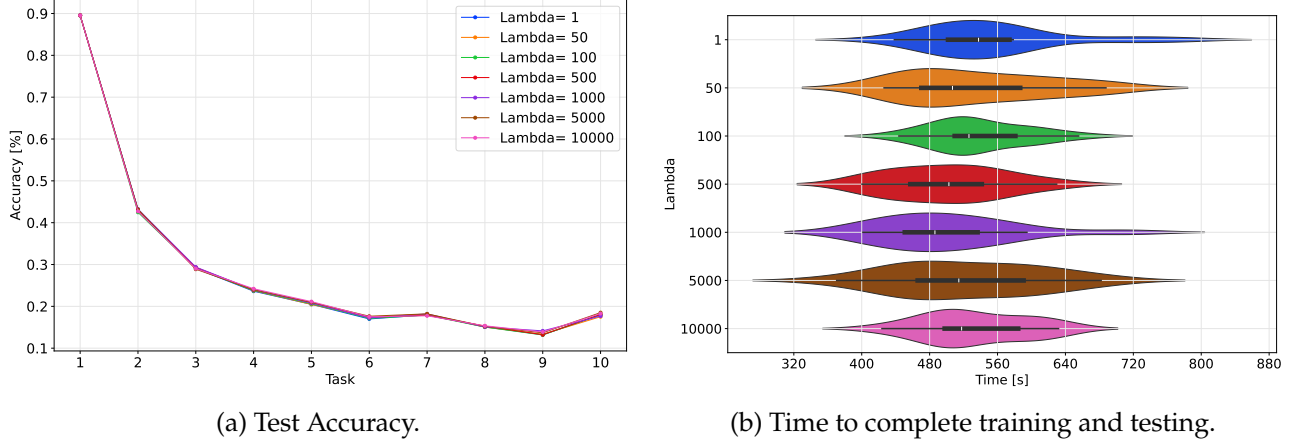


Figure A.3: Results of the λ parameter search for the EWC method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.

A.1.3 LwF

The results of the hyperparameter search for λ in the LwF method are illustrated in Figure A.4a. The outcomes can be categorized into two distinct groups: higher values, specifically 500 and 1000, resulted in a notable decline in accuracy, particularly in the later tasks. Conversely, lower values yielded marginal performance improvements, although they remained below the top-performing methods. Ultimately, we selected $\lambda = 5$, as it demonstrated the most consistent performance while slightly enhancing both accuracy and execution time compared to the other tested values.

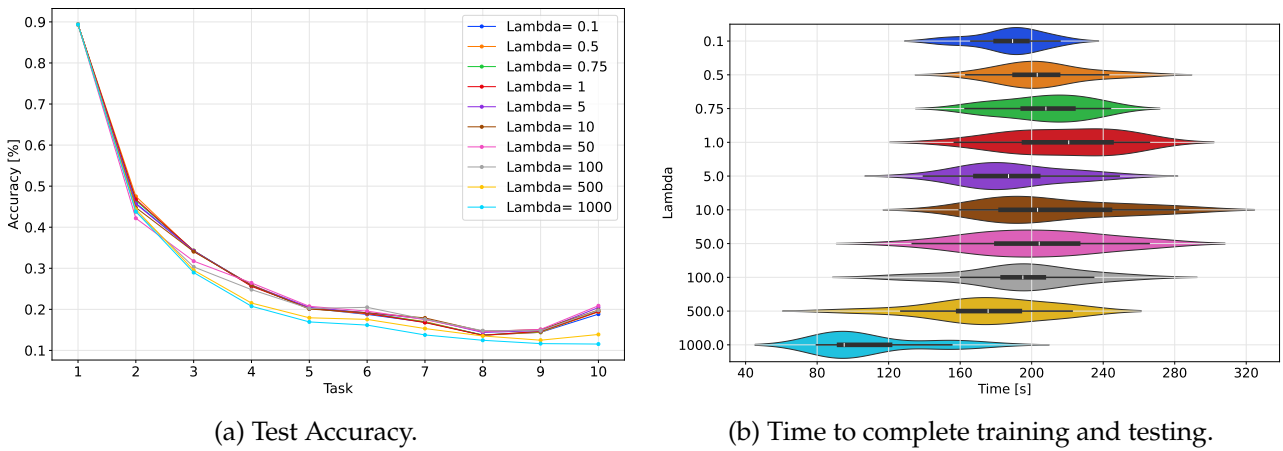


Figure A.4: Results of the λ parameter search for the LwF method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.

A.1.4 DER++

Similar to ER, the most critical hyperparameter for DER++ is the *Memory Size*. As shown in Figure A.7a, the accuracy improvement follows a logarithmic trend as *Memory Size* increases. Figure A.7b shows little to no difference in training and test time. Even if DER++ consumes slightly more memory than ER due to the storage of logits, we selected the same *Memory Size* of 1000, as it strikes the optimal balance between memory usage and accuracy gains.

Regarding the α and β hyperparameters we tried a grid of values similar to the ones reported in [238]. Figures A.5a, A.6a, A.5b, and A.6b indicate that neither has a significant impact on accuracy or training time. We observed a slight preference for setting both α and β to 1, which we adopted. This choice implies that the three loss components (Categorical Cross-Entropy on training data, MSE with respect to memory logits, and Categorical Cross-Entropy on memory labels) are treated with equal importance during model training.

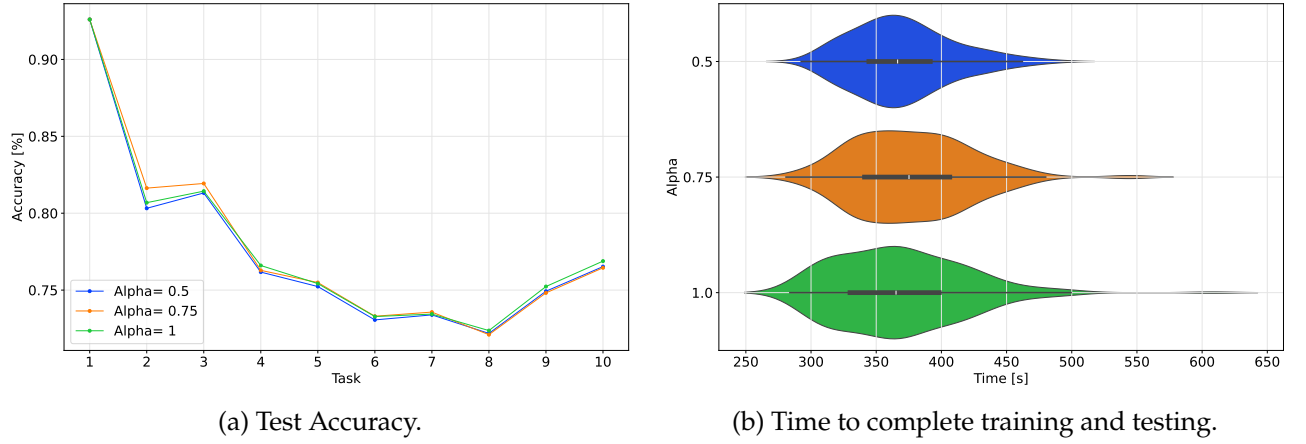


Figure A.5: Results of the α parameter search for the DER++ method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.

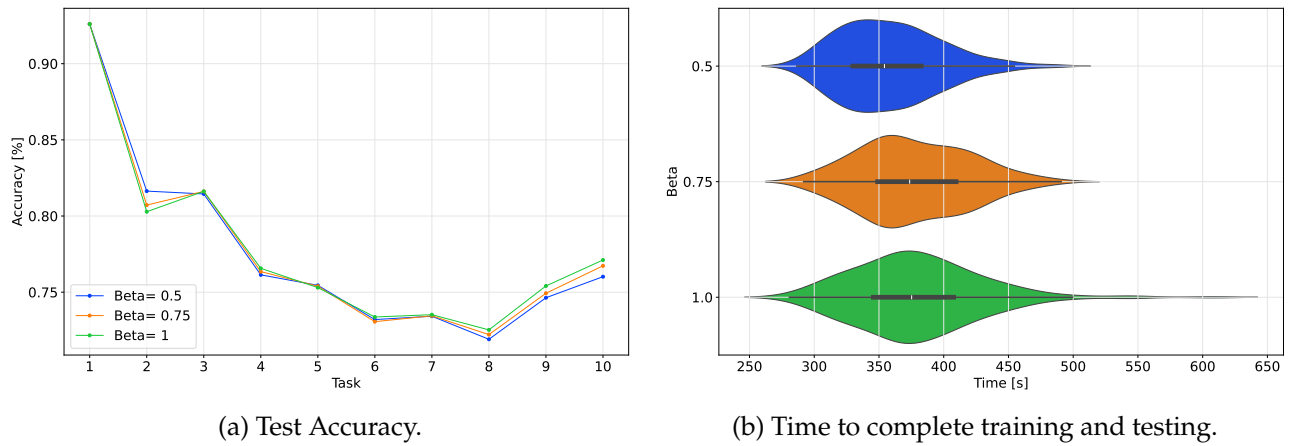
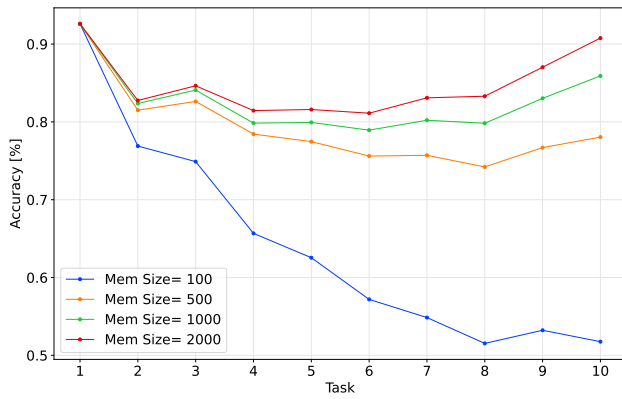
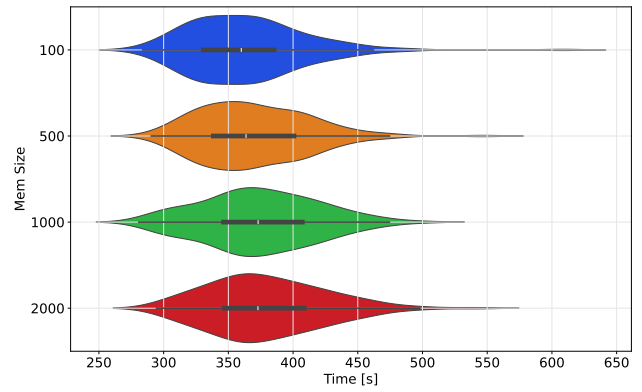


Figure A.6: Results of the β parameter search for the DER++ method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.

A.1. HYPERPARAMETERS TUNING



(a) Test Accuracy.



(b) Time to complete training and testing.

Figure A.7: Results of the *Memory Size* parameter search for the DER++ method. The test accuracy (y-axis) after training on each task (x-axis) is shown on the left for each value of the hyperparameter. On the right, a violin plot indicates the required time for training and testing.

Acknowledgments

I would like to sincerely thank my supervisors, Gian Antonio and Ruggero, for their guidance, support, and invaluable advice throughout this journey.

I am deeply grateful to ANDRITZ HYDRO for funding this doctorate, giving me the opportunity to grow both as a researcher and as a person, and for allowing me to collaborate on real-world industrial problems while sharing their profound expertise in hydroelectric systems. My heartfelt thanks go in particular to Claudio, who was always available to provide insights and practical perspectives that greatly enriched my understanding of the field, and for his support and supervision at every step of this collaboration.

Finally, I warmly thank my colleagues and friends at the University of Padova, with whom I collaborated on my work, as well as the many others who contributed to this journey or simply shared this experience with me, making it more enjoyable and inspiring.