Failure Prediction and Explanation

A Case Study in Cork Industry

Gonçalo Faria Augusto
Master in Data Science
Department of Computer Science
Faculty of Sciences of the University of Porto
2025

Failure Prediction and Explanation

A Case Study in Cork Industry

Gonçalo Faria Augusto

Dissertation carried out as part of the Master in Data Science
Department of Computer Science
2025

Supervisor
João Gama, Emeritus Professor, Faculty of Economics of the University of Porto

Co-supervisor
Rita P. Ribeiro, Assistant Professor, Faculty of Sciences of the University of Porto

“ One of the basic rules of the universe is that nothing is perfect.

Perfection simply doesn’t exist...

Without imperfection, neither you nor I would exist. ”

Stephen Hawking

Acknowledgements

I would like to express my gratitude to my supervisors João Gama and Rita P. Ribeiro for their guidance and support. Their constructive feedback has provided me the knowledge needed to make this happen.

Also, I want to thank Amorim Cork, S.A. for providing the data to develop this thesis. Their collaboration was instrumental in making this possible.

This thesis would not have been possible without the contributions and trust of everyone directly or indirectly involved.

Resumo

Os rigorosos padrões de qualidade da indústria da cortiça exigem sistemas de produção altamente fiáveis e eficientes. A grande contribuição é o desenvolvimento de um sistema de Manutenção Preditiva, em tempo real, para as máquinas de análise de cortiça NDTech 2.0, que gera dados multivariados de sensores de alta frequência, captando vários parâmetros de funcionamento (p.e., temperaturas, pressões, vibrações) em todas as suas unidades de processamento. Concentramo-nos na previsão de falhas mecânicas e operacionais graves que perturbam o controlo de qualidade, desenvolvendo uma estrutura de Manutenção Preditiva orientada por dados que permite a previsão e o diagnóstico precoce de falhas utilizando estes dados contínuos dos sensores.

Implementamos uma arquitetura de duas camadas, composta por uma Camada de Previsão de Falhas e uma Camada de Explicação. Modelos baseados em Aprendizagem Profunda, incluindo LSTM-AE e WAE-GAN, foram avaliados para detetar anomalias associadas a falhas em fase inicial. Os nossos resultados mostram que o modelo WAE-GAN supera significativamente o LSTM-AE em precisão preditiva, estabilidade arquitetónica e eficiência de treino quando aplicado às características heterogéneas dos dados das unidades de processamento NDTech 2.0. Os modelos otimizados são capazes de antecipar avarias com 30 a 70 minutos de antecedência, proporcionando uma janela crítica para intervenções de manutenção em tempo útil.

Para apoiar a tomada de decisões de manutenção, a estrutura incorpora técnicas de Inteligência Artificial Explicável utilizando LIME, que identificam as principais leituras dos sensores que contribuem para cada previsão. Estas explicações localizadas ajudam a descobrir as causas-raiz, a orientar as ações corretivas e a promover a confiança no sistema.

No geral, esta tese apresenta uma solução Manutenção Preditiva interpretável e eficaz, adaptada ao ambiente NDTech 2.0, melhorando significativamente a fiabilidade, a manutenibilidade e a eficiência operacional.

Palavras-chave: Manutenção Preditiva, Inteligência Artificial Explicável, Aprendizagem Profunda

Abstract

The cork industry's stringent quality standards demand highly reliable and efficient production systems. The main contribution is the development of a real-time Predictive Maintenance (PdM) system for cork analyzer machine NDTech 2.0, which generates high-frequency multivariate sensor data capturing various operational parameters (e.g., temperatures, pressures, vibrations) across its Processing Units (PUnits). We focus on predicting severe mechanical and operational failures that disrupt quality control, by developing a data-driven PdM framework that enables early failure prediction and diagnosis using this continuous sensor data.

We implemented a two-layered architecture comprising a Failure Prediction Layer and an Explanation Layer. *Deep Learning*-based *Autoencoder*s, including LSTM-AE and WAE-GAN, were evaluated for detecting anomalies associated with early-stage failures. Our findings show that the WAE-GAN model significantly outperforms the LSTM-AE in predictive accuracy, architectural stability, and training efficiency when applied to the heterogeneous data characteristics of the NDTech 2.0 PUnits. The optimized models are capable of anticipating failures 30–70 minutes in advance, providing a critical window for timely maintenance intervention.

To support maintenance decision-making, the framework incorporates XAI techniques using LIME, which identify key sensor readings that contribute to each prediction. These localized explanations help uncover root causes, guide corrective actions, and foster trust in the system.

Overall, this thesis presents an interpretable and effective PdM solution tailored to the NDTech 2.0 environment, significantly enhancing reliability, maintainability, and operational efficiency.

Keywords: Predictive Maintenance, Explainable Artificial Intelligence, Deep Learning

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AD *Anomaly Detection*. ix, x, 6, 10, 12, 13, 16, 18, 20, 22, 27, 29, 31, 33– 37, 43, 46, 59, 73– 76, 83, 87, 88

AE *Autoencoder*. iv, 2, 3, 9, 10, 12, 13, 27, 29, 33, 73– 75, 85

AI *Artifical Inteligence*. viii, 1, 4, 5, 7, 10– 12

ANN Artificial Neural Network. 4, 6, 9, 77

CNN *Convolutional Neural Network*. 6, 9

CPS *Cyber-Physical Systems*. 7

DL *Deep Learning*. iv, viii, 4– 7, 9, 10, 13, 16, 27, 29, 30, 33, 60, 61

DT *Decision Tree*. 77

EDA *Exploratory Data Analysis*. 19

GAN *Generative Adversarial Network*. 6, 74, 75, 86

GNNs *Graph Neural Networks*. 6, 61

GRU *Gated Recurrent Unit*. 9, 13

I4.0 *Industry 4.0*. 1, 7, 16

IoT *Internet of Things*. 7, 12

IQR *Interquartile Range*. 33

LIME *Local Interpretable Model-Agnostic Explanations*. iii, iv, vii, ix, xi, 13, 28, 30, 35, 36, 50– 60, 76– 78, 121, 122

LSTM *Long Short-Term Memory*. ix, 6, 9, 12, 13, 30, 37, 38, 73, 74, 78, 79, 81, 82, 85, 86

LSTM-AE *Long Short-Term Memory Autoencoder*. iii, iv, vii– ix, 31, 37, 38, 40– 43, 46, 59, 60, 73, 75, 76, 78, 79, 81– 83, 88, 89

MAE *Mean Absolute Error*. 9

MAPE *Mean Absolute Percentage Error*. 9

ML *Machine Learning*. viii, 2, 4, 5, 7, 10, 11, 16, 19, 20, 76, 77

MLP *Multilayer Perceptrons*. 5, 9

MSE *Mean Squared Error*. 9, 33, 37, 39, 74, 82, 83, 87, 88

NDTech *Non-Detectable Technology*. iii, iv, vii, viii, 2– 4, 15– 19, 24, 25, 29– 31, 35, 36, 50, 58– 61, 81, 84

O1 *First Objetive*. 2

O2 *Second Objetive*. 2

P-F *Potential Failure*. 7

PdM *Predictive Maintenance*. iv, viii, 1, 3, 4, 6– 14, 16, 25, 27, 29– 31, 35, 47, 50, 60, 61

PR *Precision-Recall*. 9

PUnit *Processing Unit*. iv, vii– xi, 15– 27, 29– 33, 35, 40– 60, 74, 76, 78– 80, 82, 83, 89– 122

RE *Reconstruction Error*. 28, 30, 37, 39, 40, 84, 86– 88

ReLU *Rectified Linear Unit*. 84

RF *Random Forest*. 9, 12, 32

RMSE *Root Mean Squared Error*. 9

RNNs *Recurrent Neural Networks*. 6, 73

RUL *Remaining Useful Life*. 8, 9, 13

SHAP *SHapley Additive exPlanations*. 13

SNN Spiking Neural Network. 4

SVM *Support Vector Machine*. 9, 77

TCA *2,4,6-Trichloroanisole*. 2, 15, 19

TCN *Temporal Convolutional Network*. x, 6, 12, 31, 38, 40, 75, 76, 80, 84, 85

VAE *Variational Autoencoder*. 6, 10

WAE *Wasserstein Autoencoders*. 74, 86, 87

WAE-GAN *Wasserstein Autoencoder with Generative Adversarial Network*. iii, iv, vii, viii, x, xi, 12, 31, 33, 37– 40, 43– 47, 49, 59, 60, 74– 76, 79, 80, 84, 86, 101, 117– 120

XAI *eXplainable Artificial Intelligence*. iv, viii, 1, 3, 4, 7, 11– 13, 60, 61

1. Introduction

In recent years, *Artifical Inteligence* (AI) has experienced a significant resurgence, often termed a ”new spring”, largely driven by pervasive digitization and the recognition that abundant digital data is a critical asset for deriving insights and driving business outcomes [1–4].

This evolution necessitates that companies invest in AI-underpinned technologies to remain competitive [5, 6]. The focus has shifted from immediate financial returns to understanding AI's potential for innovation, creating new business models, and enhancing long-term competitiveness [7, 8]. This long-term perspective is crucial for *Predictive Maintenance* (PdM), a key application within *Industry 4.0* (I4.0), which leverages these technologies to monitor equipment, predict failures, and optimize maintenance for significant cost savings and efficiency [9].

As AI integrates into critical decision-making, particularly in PdM, understanding and trusting AI-driven insights become paramount [10]. This aligns with the growing emphasis on ”trustworthy AI”, highlighted by initiatives like the G7's call for human-centric AI and the European Commission's ”Ethic Guidelines for Trustworthy AI”, which emphasize transparency, privacy, and data governance [11, 12]. *eXplainable Artificial Intelligence* (XAI) plays a crucial role here by making AI decision-making transparent and understandable [13]. This need for transparency, interpretability, and adaptability in dynamic industrial environments forms the core motivation for this thesis.

The current era's pervasive AI applications, combined with the profound shifts of I4.0, have revolutionized industrial production, leading to massive data generation. This vast data is analyzed to develop targeted solutions, with PdM being a key task to minimize equipment failures, reduce downtime, and lower operational expenses [14–16]. Despite its benefits, challenges hinder PdM's full adoption: most models struggle with dynamic environments, complex high-dimensional time-series data, scarcity of labeled failure data, focus on isolated components, and lack of interpretability [17]. These challenges, coupled with the need for complex reasoning and planning in real-world PdM (requiring insights for operators, technicians, and managers), motivated this thesis.

The integration of AI and I4.0 technologies has transformed manufacturing, making PdM critical for reliability and efficiency [14]. The cork industry, with Portugal as the largest

producer, has also embraced these advancements, particularly in quality control. Cork, historically valued for its unique qualities and primarily used for wine bottle closures since the late 1700s [18, 19], faces a major challenge: ”cork taint”, primarily caused by *2,4,6-Trichloroanisole* (TCA), which negatively affects wine aroma and taste [20, 21]. To address this, companies like Amorim Cork, S. A., a world leader, developed *Non-Detectable Technology* (NDTech) technology. This groundbreaking system uses non-destructive gas chromatography to precisely analyze each cork stopper, significantly reducing analysis time from 15 minutes to around 20 seconds while maintaining precision. NDTech provides a highly reliable and accurate alternative to subjective human sensory analysis, ensuring TCA-free corks and preserving wine integrity [18].

1.1. Objectives

In fact, the TCA detection process is critical for the cork industry, where guaranteeing its quality and reliability is crucial. The prevalence of advanced and complex machinery becoming the new standard in most manufacturing facilities introduces new challenges and constraints in control procedures, calling for adopting new and more efficient practices. NDTech stands as a remarkably sophisticated technology, a vivid example of this scenario as the complexity of the multi-component architecture demands close control to ensure the proper functioning of all the system elements, which currently relies on a periodic manual control procedure, making it susceptible to human error and limitations on both efficiency and resources demands. This thesis aims to identify and explore the possibility of improving the current control implemented on NDTech, where the information regarding the different variables and indicators provided by the sensors measures distinct functional parameters of the system. Moreover, building on previous efforts, this study aims to move into new unexplored areas, gather more information and knowledge regarding the data, and explore different algorithms to solve the maintenance problem. This *First Objetive* (O1) involves exploring *Machine Learning* (ML) techniques, specifically *Autoencoder* (AE), to develop a real-time automated anomaly detection system that could be implemented on-site and used by the NDTech operators. The *Second Objetive* (O2) is to produce explanations of the failures or rare events to help the NDTech operators perform more accurate maintenance tasks on the machines.

To achieve the objectives above with a focus on the expected results, the development of this work relied on the following macro stages:

- Conduct a comprehensive review of PdM and XAI in industrial contexts to establish a strong theoretical foundation.
- Develop robust mechanisms for cleaning, transforming, and engineering features from NDTech 2.0 sensor data, preparing it for model training.
- Design and implement the core system using different AEs models.
- Integrate an XAI technique with the trained models to generate human-understandable explanations for predicted failures.
- Evaluate the developed system, including its prediction accuracy and the quality of explanations, to ensure its effectiveness in providing timely and accurate insights.
- Analyze results, draw conclusions, and future research directions.

1.2. Document Structure

This thesis is organized as follows: Chapter 1, Introduction, provides the user with the main motivations and contextualization of the topic at the present time, as well as the definition of the objectives and expected results; Chapter 2, Background, focuses on providing the reader with a better understanding of research methodologies and state-of-the-art relevant to the thesis; Chapter 3, Case Study in the Cork Industry, a brief description of what this thesis is about and how it is organized in technological terms, describing the high-level view architecture; Chapter 4, Experimental Study, presents the models' architectures, and evaluation of the failure prediction models, as well as the application of interpretability technique to explain detected failures; Chapter 5, Conclusion and Future Work, gives the final notes, drawing from the same conclusions for the future. Finally, the References of all consulted sources and the Appendix with all detailed (process) information and supplemental visualizations of the results obtained.

2. Background

This chapter presents the relevant state-of-the-art. It starts by presenting the bibliographic research strategies in Section 2.1. Then, reviews the essential concepts and presents a view of the scientific community related to AI, PdM, and XAI. It surveys the theoretical foundations and also provides a state-of-the-art analysis of recent and relevant studies related to these concepts.

2.1. Bibliographic Research

This research aimed to gather foundational knowledge for the project, using a structured methodology and focusing on key technical areas.

Relevant sources (e.g., Google Scholar, ScienceDirect, IEEE Xplore) were reviewed to collect literature. From this, documents containing key terms were selected and summarized to extract the most relevant insights.

The study focused on topics essential for implementing a PdM system in the NDTech environment, including PdM, failure detection/prediction, anomalies, outliers, explainability, pattern recognition, and *Deep Learning* (DL). Emphasis was placed on understanding failure types in multi-sensor systems and leveraging ML, particularly DL, to enhance detection and support real-time, data-driven decisions. Sensor integration and data acquisition were also explored to ensure accurate modeling of system behavior.

Therefore, research focused on recent and relevant studies in PdM, DL, and XAI, using top keywords such as: predictive maintenance, explainable ai, deep learning algorithms, data-driven predictive maintenance, and anomaly detection.

2.2. Taxonomy of AI

As AI has evolved, its growing complexity has necessitated structured taxonomies to clarify relationships among its diverse subfields. Visual tools such as Euler diagrams (Fig. 2.1) effectively illustrate how key domains like ML, Artificial Neural Network (ANN), and DL overlap, while also integrating newer paradigms like Spiking Neural Network (SNN) [22].

Within AI, ML has been particularly transformative, and the development of ANN gave rise to DL, which has demonstrated exceptional performance across numerous applications

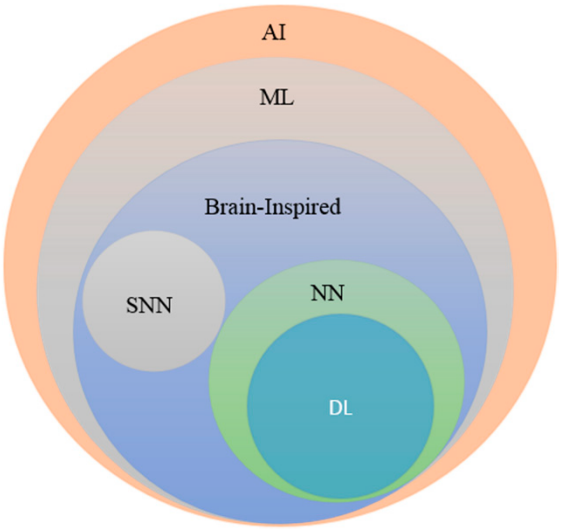


Figure 2.1: The taxonomy of AI [22].

since its emergence around 2006 [22–24]. The standard ML pipeline (Fig. 2.2) begins with raw data collection and preprocessing, followed by model training and inference.

Figure 2.2: An illustration of the pipeline of building a ML system [25].

DL, a subset of ML, distinguishes itself through its ability to automatically extract hierarchical representations of data using deep, multilayered architectures. As shown in Fig. 2.3, DL tends to outperform traditional ML when sufficient data is available [26].

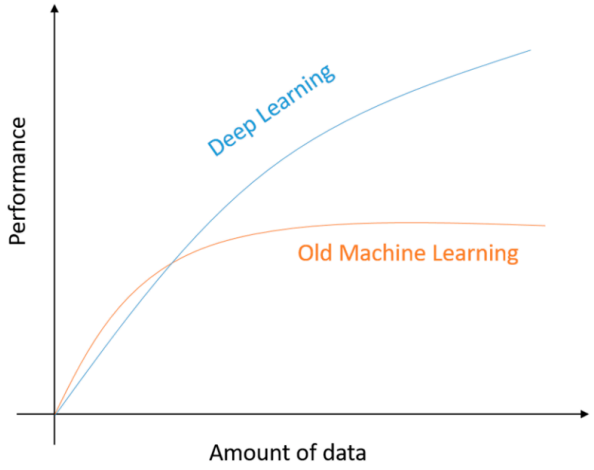


Figure 2.3: The performance of DL with respect to the amount of data [22].

Several DL architectures cater to specific data types. *Multilayer Perceptrons* (MLP) are

fully connected networks ideal for structured data [27], while *Convolutional Neural Network* (CNN) are optimized for grid-like inputs such as images and have been applied to visual inspection and time-series-to-image transformations in PdM [28–30]. *Recurrent Neural Networks* (RNNs), including *Long Short-Term Memory* (LSTM) variants, model sequential dependencies and are widely used in forecasting and *Anomaly Detection* (AD) [31–35]. Transformers, originally developed for natural language processing, leverage self-attention mechanisms for parallel sequence modeling and have shown strong performance in time-series analysis and multimodal fusion [36, 37]. *Graph Neural Networks* (GNNs) generalize ANN to graph-structured data and are effective in modeling the relational structure of industrial systems [38]. Generative models such as *Variational Autoencoder* (VAE) and *Generative Adversarial Network* (GAN) are valuable for unsupervised learning, AD, and data augmentation in failure-scarce environments [39–41]. *Temporal Convolutional Network* (TCN) apply causal convolutions to sequence data, often outperforming RNNs in time-series tasks due to stable gradients and parallelizability [42].

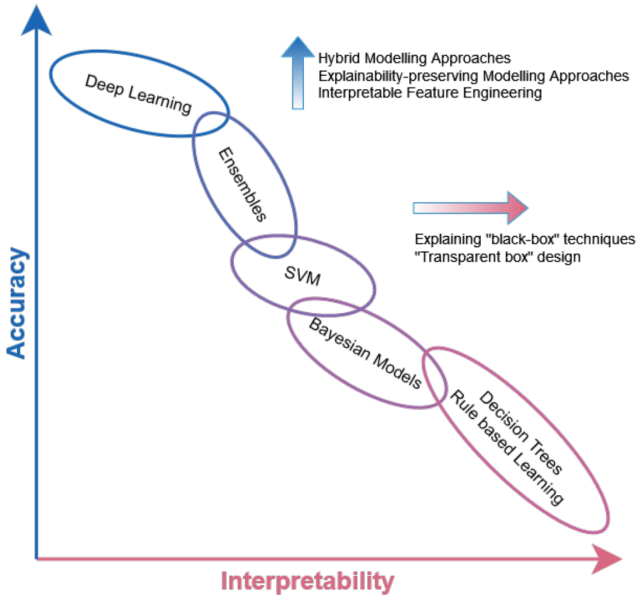


Figure 2.4: Trade-off between model interpretability and performance [10].

DL offers several advantages: automatic feature learning, the ability to handle unstructured data, scalability with large datasets, and end-to-end optimization pipelines [26, 43]. However, these strengths are accompanied by challenges. DL models typically require extensive labeled data, significant computational resources, and suffer from poor interpretability, which is critical in high-stakes domains like PdM. Fig. 2.4 illustrates the common trade-off between model performance and interpretability [10, 44]. Moreover, DL

models are sensitive to hyperparameters and prone to overfitting without adequate regularization or data. To mitigate these issues, ongoing research in XAI seeks to enhance transparency and trust in DL systems—especially vital in industrial applications where understanding predictions is essential for effective maintenance planning, as further discussed in Section 2.4.

2.3. Predictive Maintenance

PdM is a cornerstone of modern industrial operations, especially within I4.0. It leverages real-time data and advanced analytics to anticipate equipment failures, enabling just-in-time maintenance, minimizing unplanned downtime, reducing costs, and extending operational lifespan.

I4.0, introduced in Germany in 2011 [45], represents the fourth Industrial Revolution and the digital transformation of manufacturing. It delivers real-time decision-making, enhanced productivity, flexibility, and agility by connecting computers and industrial equipment to make decisions without human intervention. This revolution is powered by disruptive technologies such as the *Internet of Things* (IoT), Big Data, Cloud Computing, AI, ML, Edge computing, Digital Twin, and *Cyber-Physical Systems* (CPS), necessitating increased investment to maintain competitiveness. A key concept is CPS, integrating computational systems with physical processes for coordinated and efficient interaction.

Implementing I4.0 technologies introduces maintenance strategies to prolong equipment lifespan and cut costs through continuous communication. PdM is vital here, using these technologies to anticipate and prevent breakdowns, reducing costs and minimizing downtimes. Industrial maintenance has evolved significantly, from merely addressing breakdowns (reactive/corrective maintenance, common until after WWII) to planned preventive maintenance (emerging post-1950s to reduce losses) and, more recently, to PdM, also known as smart maintenance. This evolution, depicted in Fig. 2.5, shifts away from expensive reactive approaches. PdM provides the longest equipment lifespan, highest reliability, and most cost-effective solutions. Proactive maintenance, focusing on root cause analysis, is gaining popularity when used alongside PdM.

Equipment failure is nearly random. A key technique for maintenance decision-making is *Potential Failure* (P-F) curve analysis (Fig. 2.6), which illustrates the time between potential and actual failure detection.

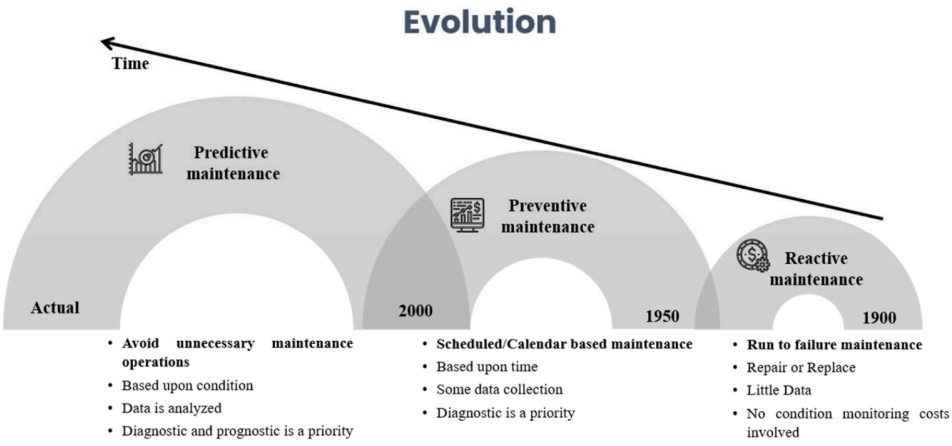


Figure 2.5: Evolution of maintenance activities and methods [46].

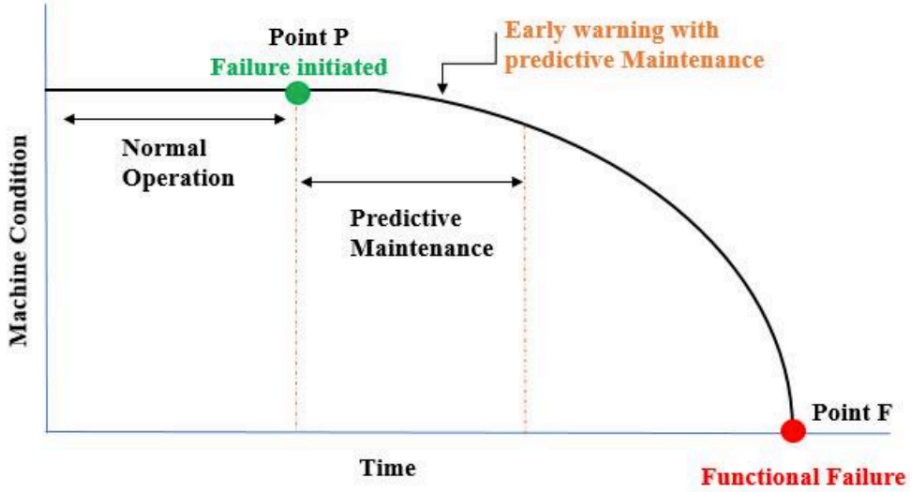


Figure 2.6: P-F diagram presenting inspection intervals and PdM [46].

PdM practices are developed from two perspectives (Fig. 2.7): failure prediction (forecasting approximate failure time) and *Remaining Useful Life* (RUL) estimation (predicting operational time before repairs are needed).

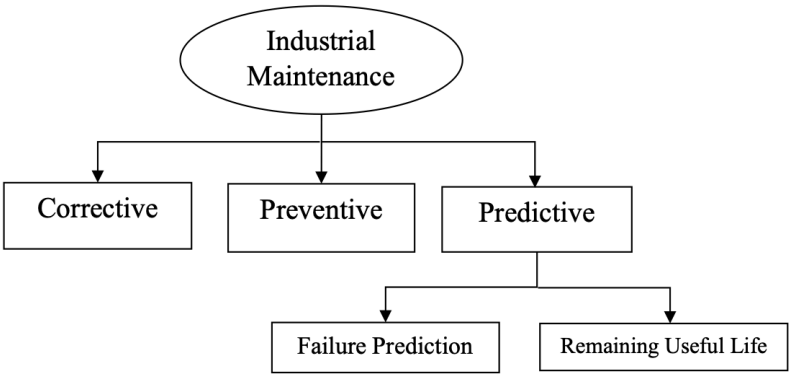


Figure 2.7: Classification of automatic industrial maintenance approaches [47].

Despite its inevitability, adopting PdM faces challenges, primarily balancing investment

costs (sensors, installation, consultancy) with expected returns. Another significant challenge is the need for accurate and relevant data; incomplete or inaccurate data can lead to erroneous predictions.

Evaluation of failure prediction methods primarily uses Accuracy and *Precision-Recall* (PR) score. RUL prediction methods are assessed using *Mean Absolute Error* (MAE), *Mean Absolute Percentage Error* (MAPE), *Mean Squared Error* (MSE), and *Root Mean Squared Error* (RMSE). For abnormal cycle detection, False Alarm Rate (FAR) and Impostor Pass Rate (IPR) are used, with rFAR and rIPR offering enhanced outlier detection and early failure prediction.

Data-driven PdM methods involve three general steps: data acquisition, data preprocessing, and model building. Data acquisition, the initial stage, gathers industrial data via sensor connections (most common, capturing vibration, temperature, pressure), external capture, inspection logs, or data simulation. Data preprocessing enhances prediction quality through techniques like handling missing values, data summarization, and feature selection, often condensing features into a lower-dimensional space for efficient real-time processing. Finally, the model building phase applies data analysis to identify anomalies, failures, or RUL predictions, using various learning methods (supervised, semi-supervised, unsupervised) based on data characteristics and task requirements.

Supervised learning In classification, discrete labels estimate health status by categorizing data into different failure types, including binary, multiclass, or multilabel scenarios. Binary classification distinguishes between two states (e.g., correct/malfunction), with DL classifiers like LSTM and *Gated Recurrent Unit* (GRU) often outperforming traditional methods (KNN, *Support Vector Machine* (SVM), *Random Forest* (RF), ANN). Multiclass classification handles distinct failure modes, where algorithms like RF and ANN have shown strong performance, sometimes in hybrid architectures combining CNN with ELM or stacked LSTM layers. Regression, conversely, predicts continuous variables like health indices or RUL. Ensemble learning algorithms like RF, GTB, and XGB often outperform individual MLP Regressor and SVR. ARIMA approaches, individually or combined with SVM, and Hybrid Deep Neural Networks (HDNNs) integrating LSTM and CNN, are also used for RUL estimation. AE models composed of stacked LSTM units are also used for time-series reconstruction to evaluate machine condition and predict RUL.

Unsupervised learning This approach segments data into operational modes using clustering and identifies unusual patterns indicative of anomalous behavior. In data mining, clustering groups similar items, with K-Means being a prominent partitional clustering algorithm. Newer methods like K-Multi-Dimensional Time-Series Clustering (K-MDTSC) are designed for multivariate time-series. Streaming-based clustering algorithms such as CluStream, ClusTree, and StreamKM are used for real-time AD, where an increase in outliers correlates with approaching failure. Density-based clustering, like DBSCAN (and its stream-adapted version DenStream for real-time data), identifies densely populated areas and anomalies, bridging clustering and outlier detection. DL models, such as those based on Hierarchical Temporal Memory (HTM), are also applied for efficient online outlier detection in real-time vibration data by learning and predicting patterns in time-series data.

Semi-supervised learning One-class classification builds a predictive model using only normal operation data to detect malfunctions, acting as novelty detection. Common methods include One-Class Support Vector Machine (OCSVM) with an RBF kernel and AEs. AEs learn normal data characteristics, and discrepancies in reconstruction errors for new data indicate anomalies. For instance, in train door breakdown prediction, AEs and OCSVM effectively reduced false alarms by differentiating structural failures from user interference. Generative classification combines a small proportion of labeled data with a large amount of unlabeled data assumed to belong to known labels. This approach, like VAE based deep generative models for bearing fault diagnosis, learns from labeled data while generating labels for unlabeled data, enhancing generalization.

2.4. eXplainable AI

The rise of advanced AI and ML techniques, particularly DL architectures, has significantly enhanced predictive performance across numerous domains [24, 26]. In industrial applications such as PdM, these models often outperform traditional methods by capturing complex non-linear patterns in large datasets. However, their opaque nature as ”black-box” systems presents major challenges [48], especially in safety-critical environments where understanding the rationale behind a prediction is essential for trust, debugging, regulatory compliance, and decision-making [49–51]. Without insight into why a failure

is predicted, these systems risk reduced adoption, safety concerns, and economic impact. Therefore, interpretability and transparency have become central to responsible AI development [52].

XAI has emerged to address this lack of transparency by developing methods that help users interpret and trust AI outputs [53]. It aims to answer essential questions such as ”why was a particular prediction made?”, ”how was it derived?”, and ”under what conditions might it fail?” [48–50], thereby enhancing trust, transparency, and user confidence (Fig. 2.8).

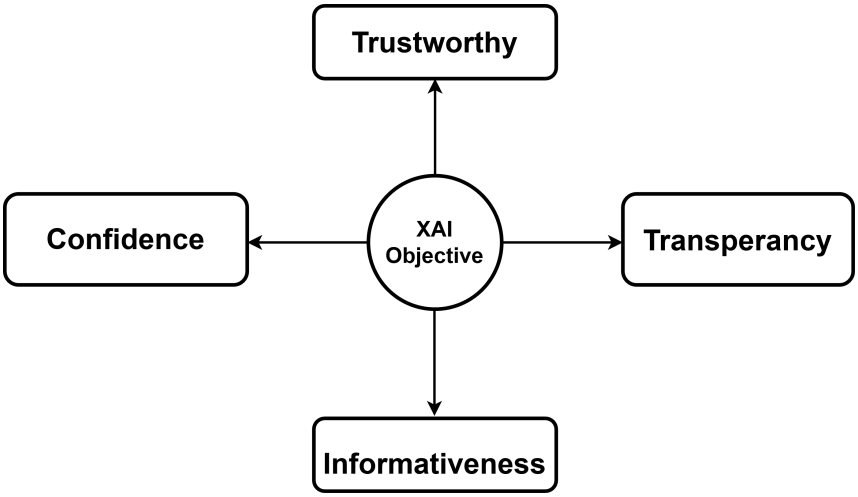


Figure 2.8: Objectives of XAI [54].

XAI techniques make the internal logic of complex models comprehensible to various stakeholders—from engineers to regulators—by clarifying how specific inputs influence predictions. In [48] the authors define XAI as a set of methods that allow humans to understand and trust ML outputs. They emphasize its role in improving debugging, robustness, fairness, compliance, and user acceptance, particularly in high-stakes domains like healthcare and industrial automation.

Transparency enables users to evaluate how models function and detect biases or vulnerabilities that may compromise fairness or safety. Clear explanations also increase trust, especially when users can verify that predictions are based on logical reasoning. This is vital in critical tasks, where trust in automated systems directly affects adoption and reliance.

Interpretability in XAI spans several dimensions. ”Local” interpretability focuses on explaining individual predictions—crucial in PdM to understand isolated alarms [55, 56]—while ”global” interpretability aims to reveal how models operate across all inputs [50, 57].

Another distinction lies between ”ante-hoc” (inherently interpretable) models, such as decision trees and linear models, and ”post-hoc” techniques that explain opaque systems after training [48]. Though ante-hoc models offer transparency by design, they often lack the performance needed for complex industrial data. Consequently, much of modern XAI research focuses on post-hoc methods, including feature attribution, saliency maps, rule extraction, and counterfactual explanations. These are tailored to audiences ranging from AI researchers to maintenance engineers and lay users [58], enabling informed, human-centered interaction with complex predictive systems.

2.5. Related Work

The integration of PdM and XAI has been widely investigated across various industries, including transportation, manufacturing, and energy. These studies emphasize early failure prediction and transparent diagnostics, key themes also explored in this thesis on cork industry applications.

In transportation, Najjar et al. [59] developed a PdM framework for metro Air Production Units using real-time analog and digital sensor data, achieving high F1-Scores with a RF model. Similarly, Davari et al. [60] employed a sparse AE for AD, while Wang et al. [61] leveraged an LSTM-based AD model with adaptive thresholds and IoT-cloud integration, improving maintenance outcomes in real metro systems.

Interpretability remains a key challenge in such complex settings. Neuro-symbolic approaches, such as those proposed in [62, 63], combine deep learning with rule-based systems to provide explanations for anomalies, enhancing actionable insights. These were deployed on real metro data, with components like rule-learning modules offering sensor-level explanations. Similarly, Silva et al. [64] applied a *Wasserstein Autoencoder with Generative Adversarial Network* (WAE-GAN) with an explainability layer for early fault detection.

More broadly, Li and Jung [65] reviewed anomaly types (point, interval, series) and favored LSTM and AE-based approaches, while stressing the need for XAI. He and Zhao [66] demonstrated how TCNs detect anomalies via deviation between predicted and actual values, showing robustness across temporal scales. Han et al. [67] presented ADBench, a benchmark suite evaluating 30 AD algorithms under diverse noise and supervision levels, aiding model comparison and generalization. For real-time adaptation, Esteban Toscano [17] introduced incremental decision trees (e.g., MIHT, MLHAT), achieving

online degradation detection and fault prediction.

Other industrial examples of PdM-XAI integration include [68], where *SHapley Additive exPlanations* (SHAP) was used to explain sensor contributions in tree-based predictive models. Federated and privacy-preserving approaches were made interpretable via *Local Interpretable Model-Agnostic Explanations* (LIME) and SHAP in [69], while [70] used saliency maps for diagnosing press machinery failures. In turbofan engines, LSTM and SHAP were combined in [71], with similar XAI integration into digital twins for RUL prediction in [72]. Applications in aviation datasets were also covered in [73], using LIME to interpret GRU-based models.

Rule-based interpretability for real-time metro failure detection was developed in [74], while agricultural pumps were monitored using SHAP and counterfactuals in [75]. Knowledge graphs supported failure reasoning in wind turbines via XAI4Wind [76], and time-series explanation for marine engine failures was tackled using SHAP in [77]. In semiconductor contexts, multi-fault wafer classification was interpreted using SHAP and LIME [78], and pump bearing failures were explained using LRP-applied LSTMs in [79].

Collectively, these studies showcase how predictive modeling and interpretability methods work hand-in-hand to deliver robust, early-warning systems—insights critical for developing fault explanation and early failure prediction models in the cork industry.

2.6. Discussion

While prior work in PdM has shown the potential of DL models (such as in AEs family) for failure prediction and of XAI techniques for interpretability, most approaches remain limited in scope. They often rely on simplified or simulated environments, focus on general-purpose equipment, and lack integration with domain-specific operational constraints.

Furthermore, existing methods typically separate failure detection from failure explanation, making it difficult to derive actionable insights in real time. This disconnect reduces their utility in high-precision manufacturing settings, where early, explainable interventions are critical to maintaining quality and throughput.

This thesis builds on those foundations but shifts the focus to a real-world industrial context with complex, heterogeneous sensor data. It explores how advanced AD methods and localized interpretability techniques can be adapted and optimized to deliver timely, trustworthy predictions in a production-critical environment.

By addressing these challenges, the work contributes a practical, interpretable PdM solution that aligns more closely with the operational demands and reliability standards of modern manufacturing.

3. Case Study in the Cork Industry

This chapter lays out the approach taken to build our system for the NDTech 2.0. It will begin by describing the problem, then proceed to the tools and environment used, and examine the data itself, including how it was prepared. Following that, it will be detailed the core implementation of the system, covering how it is detected potential failures and how the explanations are generated. Finally, it will be explained how the models were trained and tested, and how their success was measured.

3.1. Problem Description

As previously discussed, TCA detection is paramount in the cork industry due to its direct impact on product quality. Modern manufacturing, with its sophisticated machinery and automation, brings new challenges to quality control. The NDTech system, particularly its second generation (NDTech 2.0), exemplifies this: its intricate, multi-layered architecture demands constant oversight for optimal component operation.

Currently, NDTech 2.0's quality control relies on manual periodic checks. While functional, this approach is inherently limited by human error, time inefficiencies, and high resource consumption. This project, therefore, emerged from the need to modernize monitoring by leveraging the extensive operational data already generated by the system's sensors.

NDTech 2.0 is a gas chromatography-based technology designed for individual cork stopper analysis, capable of identifying TCA concentrations below 0.5 ng/L in a rapid 20-second cycle. The system operates in three primary phases: sample preparation, chromatographic separation, and TCA detection.

Each NDTech 2.0 *Processing Unit* (PUnit) comprises four autonomous processing modules sharing a centralized cork feeding mechanism, directed by a robotic transport line. The basic structure of an NDTech 2.0 PUnit is illustrated in Fig. 3.1.

Cork stoppers pass through pre-heating and incubator chambers (`Preheater_Temp`, `Chamber_Temp`) to prepare samples. Gas samples are then precisely routed and managed by a series of critical valves—the Stream Selector (`SS_Position`, `SS_Torque`), Load Inject (`LI_Position`, `LI_Torque`), Column Selector (`CS_Position`, `CS_Torque`), and Detector Selector (`DS_Position`,

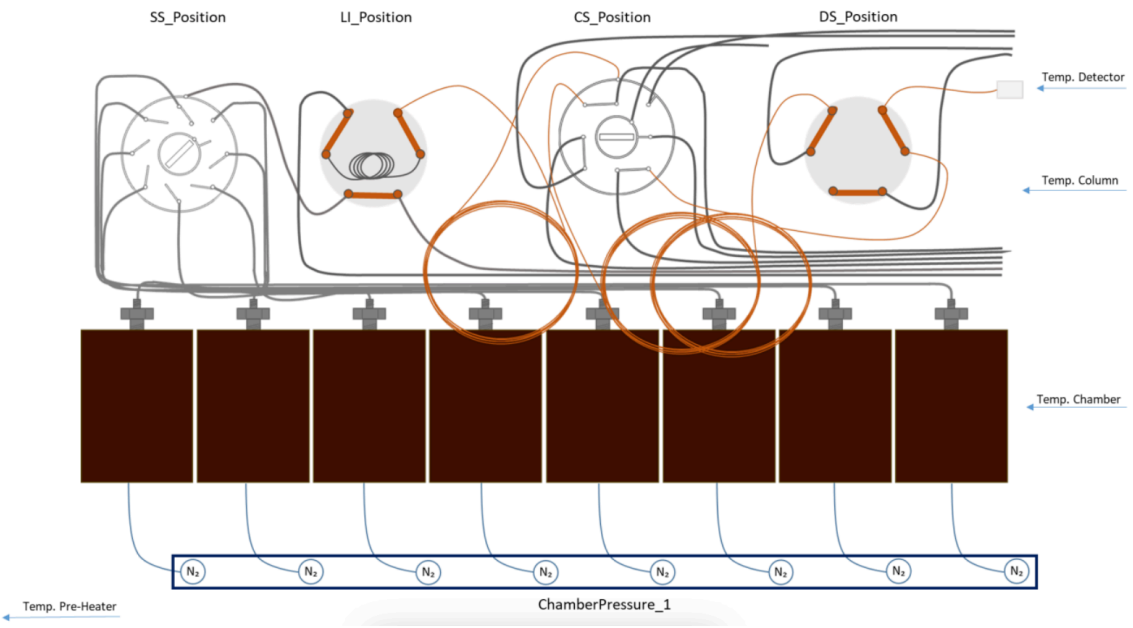


Figure 3.1: General layout of a PUnit within NDTech 2.0 system.

`DS_Torque`) valves. These components, supported by multiple nitrogen lines (Box104-Box108), perform functions like gas routing, volume control, and chemical separation, ensuring only relevant compounds reach the Electron Capture Detector (ECD).

While the existing manual procedure offers basic integrity, the increasing complexity and high-speed demands of NDTech 2.0 highlight its limitations. PdM offers a promising alternative: by exploiting the continuous stream of data from embedded sensors, it enables early identification of abnormal patterns. This aligns with I4.0 principles, making data-driven maintenance central to system resilience [9, 15, 46].

Applying PdM to the NDTech system allows for anticipating degradation, optimizing maintenance, reducing downtime, and sustaining high-throughput quality control. Given the volume and complexity of sensor data, integrating ML and DL models is essential for extracting insights and supporting real-time AD. This thesis explores a data-driven methodology to enhance NDTech 2.0's operational reliability and reduce human supervision dependence.

3.2. Dataset Description

The dataset utilized in this study was collected from the NDTech 2.0 system and reflects detailed operational data generated during each cork stopper analysis cycle. The system produces one reading every 100 milliseconds throughout a full 20-second cycle, resulting in approximately 200 high-frequency measurements per cork. To manage the resulting

volume of data and control storage costs and processing demands, only aggregated values are stored in the database, specifically, one summarized observation per cork.

This study leverages two distinct datasets collected from the NDTech 2.0 system, each serving a different purpose in the failure prediction pipeline.

- **Training dataset:** Utilizing a substantial dataset comprising 10 days of historical operational data between November 24th and December 3rd, 2024, comprising 131,067 records, confirmed to represent exclusively normal operating conditions. This dataset is used for offline training of unsupervised failure prediction models that will detailed in the Section 3.4.4.1.
- **Test dataset:** Collected throughout January 2025, comprising 459,707 records. This dataset reflects the full production environment and is used for model evaluation and online failure prediction.

Both datasets share the same structure, with each record corresponding to a full cork stopper analysis cycle and containing 59 operational variables derived from sensors monitoring mechanical, thermal, and flow-related parameters across multiple components of the system.

For continuous sensor measurements, such as torque, temperature, pressure, or valve positions, the dataset stores the average value per cork, and in some cases, also includes the minimum and maximum values observed throughout the 20-second analysis cycle. In addition to these, a small number of categorical variables are included to identify the PUnit and chamber involved in each operation. Table 3.1 provides a detailed listing of all recorded features used in the analysis.

Structurally, the dataset includes data from four distinct PUnits (denoted by the variable `PUnit`), each representing an independent NDTech 2.0 machine. Each of these machines contains eight chambers (identified by the variable `Chamber`), responsible for analyzing individual corks during the detection process. The analyses conducted by each chamber are entirely independent, allowing for a granular view of the system's operational behavior and facilitating the localization of potential anomalies within specific modules.

A key challenge is that the NDTech system does not include an integrated mechanism for real-time labeling of operational states. As such, the dataset is unlabelled; it does not identify whether each observation corresponds to normal or faulty behavior. This limitation

Table 3.1: List of features used in the NDTech 2.0 dataset.

Feature Category	**Features**
Continuous Features	LI_Torque_Min, LI_Torque_Avg, LI_Torque_Max, SS_Torque_Min, SS_Torque_Avg, SS_Torque_Max, DS_Torque_Min, DS_Torque_Avg, DS_Torque_Max, CS_Torque_Min, CS_Torque_Avg, CS_Torque_Max, Box104_OutY_Min, Box104_OutY_Max, Box104_OutY_Avg, Box104_EffX_Min, Box104_EffX_Max, Box104_EffX_Avg, Box105_OutY_Min, Box105_OutY_Max, Box105_OutY_Avg, Box105_EffX_Min, Box105_EffX_Max, Box105_EffX_Avg, Box106_OutY_Min, Box106_OutY_Max, Box106_OutY_Avg, Box106_EffX_Min, Box106_EffX_Max, Box106_EffX_Avg, Box107_OutY_Min, Box107_OutY_Max, Box107_OutY_Avg, Box107_EffX_Min, Box107_EffX_Max, Box107_EffX_Avg, Box108_OutY_Min, Box108_OutY_Max, Box108_OutY_Avg, Box108_EffX_Min, Box108_EffX_Max, Box108_EffX_Avg, ChamberPressure_1_Min, ChamberPressure_1_Max, ChamberPressure_1_Avg, ChamberPressure_2_Min, ChamberPressure_2_Max, ChamberPressure_2_Avg, Chamber_Temp, Detector_Temp, Column_Temp, Preheater_Temp, SS_Position, CS_Position, DS_Position, LI_Position
Categorical Features	PUnit, Chamber

necessitates the use of unsupervised learning techniques for AD, which requires careful statistical and temporal analysis to identify irregularities and detect deviations from normal patterns.

To support the development and evaluation of the models deployed, failure data was provided by the maintenance team. This dataset consists of manually reported failure events for the four PUnits, indicating both the date of the failure and its severity classification (severe or non-severe).

The Tables 3.2-3.5 summarize the failure reports per PUnit. Each row corresponds to a failure occurrence, with the associated date and severity. These reports were later used as references when validating model predictions.

Table 3.2: Failure Reports - PUnit 3

Date	**Severity**
2025-01-07	Severe
2025-01-10	Severe
2023-01-13	Non-Severe
2023-01-15	Non-Severe
2023-01-16	Severe
2023-01-19	Non-Severe

Table 3.3: Failure Reports - PUnit 10

Date	**Severity**
2025-01-07	Severe
2025-01-08	Non-Severe
2023-01-10	Severe
2023-01-13	Severe
2023-01-18	Non-Severe

Table 3.4: Failure Reports - PUnit 12

Date	Severity
2025-01-07	Non-Severe
2023-01-14	Severe
2023-01-18	Non-Severe
2023-01-19	Non-Severe

Table 3.5: Failure Reports - PUnit 15

Date	Severity
2025-01-07	Non-Severe
2023-01-08	Severe
2023-01-10	Non-Severe
2023-01-12	Severe
2023-01-18	Non-Severe

3.3. Exploratory Data Analysis

Given the lack of labelled data and the complex nature of the underlying physical system, the goal of this phase is to uncover relevant patterns, identify potential sources of variability, and assess the representativeness and quality of the data collected. This section focuses on the January 2025 dataset, as it represents the full operational context where anomalies may have occurred. The training dataset (November 2024), confirmed to be normal, is not subjected to the same exploratory process, as it is assumed to reflect the system's typical behavior for model training purposes. However, as mentioned earlier, this will be detailed in Section 3.4.4.1.

As said before, these data originate from the NDTech 2.0 system and capture operational metrics from four different machines, each identified by a unique PUnit code. Each machine is equipped with eight independent analysis chambers, which conduct individual chromatographic analyses to detect the presence of TCA in cork stoppers. As each chamber operates autonomously, the dataset contains observations that reflect parallel and independent measurement cycles.

This phase of the analysis seeks to characterize the statistical properties of the variables available, explore their distribution, examine correlations, and detect potential anomalies or inconsistencies. Special attention is paid to groups of features that describe mechanical movements (e.g., valve positions and torques), thermal behavior (e.g., component temperatures), and flow characteristics (e.g., pressure and gas flow measurements), as well as timing-related variables from the chromatographic process.

3.3.1 Data Cleaning and Transformation

The initial *Exploratory Data Analysis* (EDA) phase rigorously assessed data quality for consistency. This revealed 150 missing values (`NULL`) in the `SS_Torque_Min` feature, which were addressed to prevent ML algorithm errors or biased results.

Data validation also identified an invalid `Punit` value (`-1`), correcting it to conform to valid entries (`3`, `10`, `12`, `15`), and ensured chamber identifiers were within their operational range (1 to 8). These initial cleaning steps prepared the dataset for feature transformation.

Continuous variables were standardized using `StandardScaler`*, transforming data to a mean of 0 and standard deviation of 1. This crucial step prevents features with larger numerical ranges from disproportionately influencing ML algorithms, ensuring proportional contribution and improving model performance.

In parallel, categorical chamber data (1 to 8) were transformed using `OneHotEncoder`†. This method creates binary columns for each category, preventing ML algorithms from misinterpreting nominal data (where no ordinal relationship exists) and ensuring accurate representation of distinct chambers.

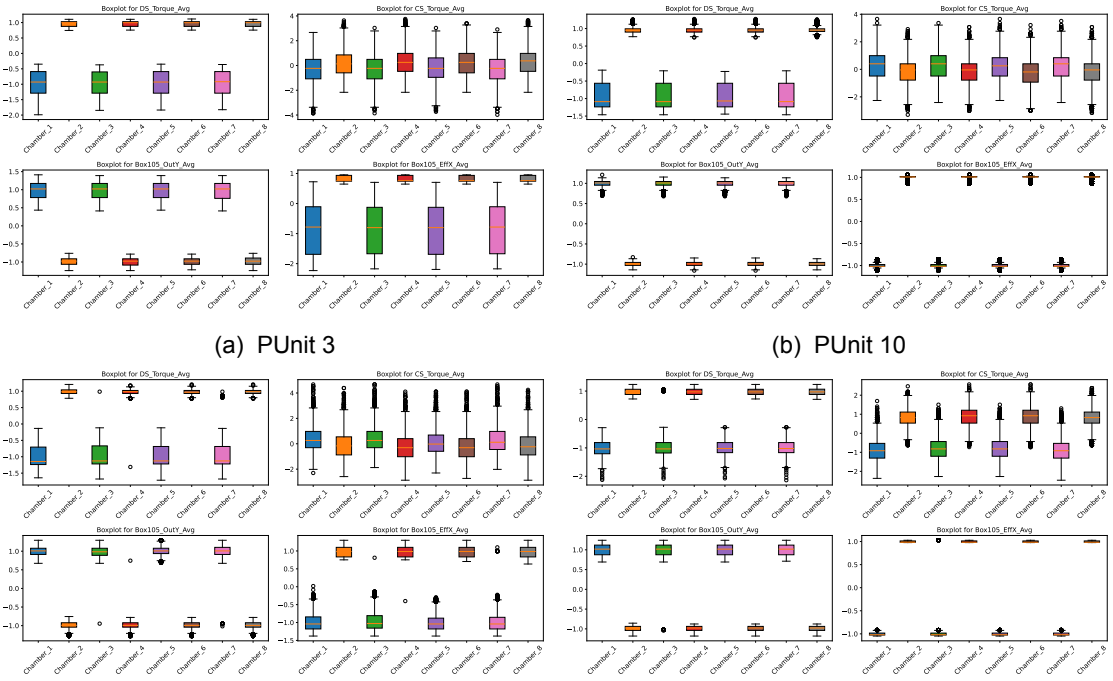
Following cleaning and transformation, the dataset was logically grouped by PUnit. This strategic step supports subsequent comparative analysis, aiding in identifying unique operational characteristics and developing PUnit-specific maintenance strategies or AD thresholds.

3.3.2 Data Visualization

The next focus is understanding the data's representativeness through the utilization of various visualization tools such as box plots, violin plots, and scatter plots, which can aid in achieving the desired objectives. These core goals center around gaining insights into the data's distribution, the differences between the available features, identifying potential subgroups within the data, detecting possible unusual values, and recognizing potential patterns and relationships among variables. Given the diverse nature of measurements represented by various sets of features, differences in how the data is characterized and distributed were anticipated. For instance, mechanical features like valve torques and positions might exhibit distinct distributional properties compared to thermal measurements or gas flow parameters.

Fig. 3.2 presents boxplots of a subset of the continuous variables, grouped by the PUnit (`PUnit`). Each subplot corresponds to one of the four PUnit (3, 10, 12, and 15), and within each subplot, the distribution of some key continuous features across the eight chambers of that PUnit can be observed.

*https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
†https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html



(a) PUnit 3

(b) PUnit 10

(c) PUnit 12

(d) PUnit 15

Figure 3.2: Boxplots of continuos variables by PUnit.

Analyzing the boxplots reveals several key observations. Firstly, for many variables, there appears to be noticeable variability in the distributions across the different chambers within the same PUnit. For example, in PUnit 3 (Fig. 3.2a), the average DS torque (`DS_Torque_Avg`) demonstrates relatively consistent behavior across chambers, while the average CS torque (`CS_Torque_Avg`) exhibits greater variation, particularly with `Chamber_2` showing a lower median and a wider spread. Furthermore, comparing the same variable across different PUnits reveals potential differences in their operational characteristics. For instance, the average CS torque (`CS_Torque_Avg`) in PUnit 10 (Fig. 3.2b) generally shows higher values compared to PUnit 3. Similarly, the `Box105_OutY_Avg` variable appears to have a broader range and different median values across the four PUnits.

The boxplots also highlight the presence of potential irregular entries in several variables and across different chambers. These are represented by individual points lying outside the whiskers of the boxplots. For example, the `Box105_EffX_Avg` in several chambers of PUnit 10 shows irregular entries, indicating instances where the measured effective flow deviates significantly from the typical range. Moreover, it's evident that the scale and range of values differ considerably across the various features. Torque values generally fall within a smaller range compared to some of the gas flow measurements (e.g., `Box108_EffX_Avg`). This difference in scale will likely need to be addressed during the modeling phase, potentially through standardization or normalization techniques.

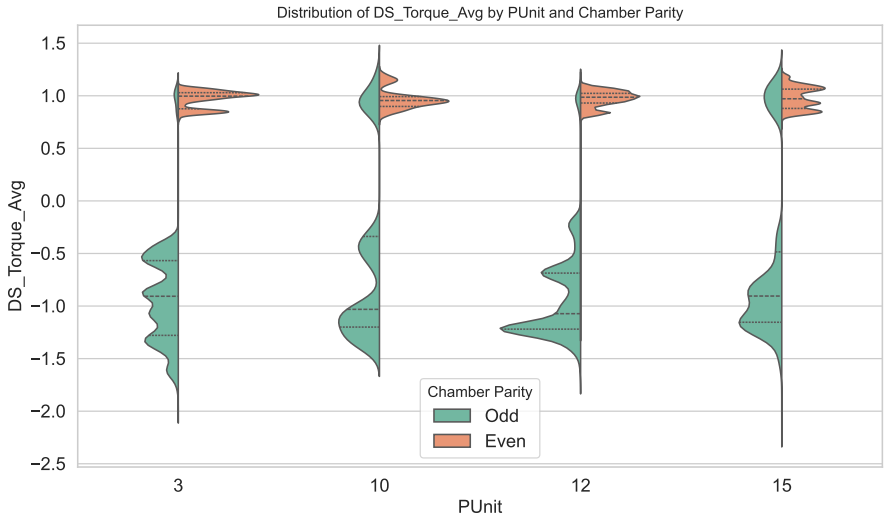


Figure 3.3: Distribution of `DS_Torque_Avg` by PUnit and Chamber Parity.

Interestingly, for some variables, a distinct pattern emerges where the distributions of even-numbered chambers tend to exhibit similar characteristics to each other, as do the distributions of odd-numbered chambers, although a systematic difference might exist between these two groups. This pattern suggests a potential influence of the physical structure or the operational sequencing of the system on the measurements. To further illustrate this, a split violin plot showing the distribution of `DS_Torque_Avg` by `PUnit` and Chamber Parity (Odd vs. Even) is presented in Fig. 3.3. As depicted in this plot, particularly when observing the `DS_Torque_Avg` across the various PUnits, the distributions for odd-numbered chambers (1, 3, 5, 7) consistently cluster together, often with a different median and spread compared to the even-numbered chambers (2, 4, 6, 8), which form another distinct cluster. For instance, when observing the `DS_Torque_Avg` in PUnit 12 (also visible in Fig. 3.2c, though less explicitly showing parity), the odd-numbered chambers show a narrower, more concentrated distribution for `DS_Torque_Avg` at a higher value, contrasting with the even-numbered chambers which exhibit a wider spread and a lower central tendency. This visual evidence strongly supports the hypothesis of a standardized operational behavior based on chamber parity, providing valuable insights into the system's internal workings. This type of inherent standardization could be highly relevant for AD, as any deviation that disrupts this established parity pattern would be a strong indicator of abnormal system behavior.

Further detailed examination of the operational data reveals consistent patterns in the distribution of torque variables, significantly influenced by both the specific PUnit and the parity of the analytical chamber. Fig. 3.4 presents a series of stacked histograms for

the average torque of key valves (`LI_Torque_Avg`, `SS_Torque_Avg`, `DS_Torque_Avg`, and `CS_Torque_Avg`) organized by `PUnit` and further segmented by Chamber Parity (Odd or Even).

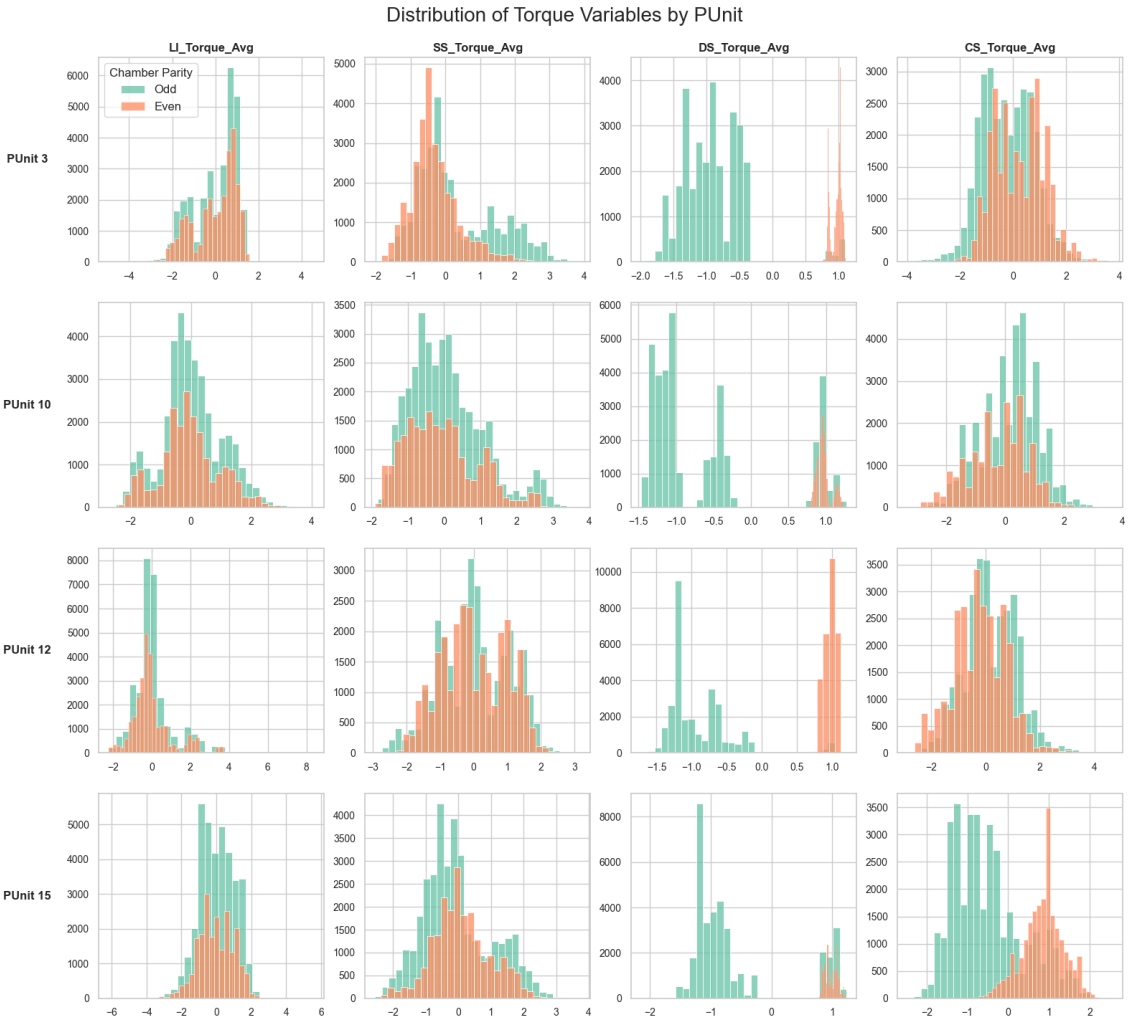


Figure 3.4: Distribution of Torque Variables by PUnit.

Across all PUnits, a striking bimodal or multi-modal distribution is frequently observed for `LI_Torque_Avg` and `CS_Torque_Avg`, and this characteristic is strongly tied to chamber parity. For instance, in PUnit 3, `LI_Torque_Avg` clearly shows two distinct peaks, with the odd chambers contributing predominantly to one peak and even chambers to the other. This parity-based segregation is consistently visible for `LI_Torque_Avg` and `CS_Torque_Avg` across all presented PUnits, suggesting that the mechanical operations of these valves inherently differ based on whether the cork is processed in an odd or even-numbered chamber. This could imply distinct operational profiles or slightly different physical configurations for these two groups of chambers.

In contrast, the DS_Torque_Avg distributions exhibit a more unique and often less bimodal pattern based on parity. While DS_Torque_Avg for odd chambers generally forms a tight, unimodal distribution at a higher value (around 0.5 to 1.0), the even chambers often display a broader or more dispersed distribution at lower values (often around -1.0 to -0.5), as particularly evident in PUnits 3, 10, and 15. This suggests a more pronounced and consistent difference in the DS_Torque_Avg behavior between odd and even chambers compared to LI and CS torques.

The SS_Torque_Avg also shows interesting variations. While some PUnits like 3 display a somewhat bimodal distribution related to parity, others, such as PUnit 12, appear to have more overlapping or less distinctly separated distributions for odd and even chambers, though still with noticeable differences in their central tendencies or spreads. These visualizations reinforce the notion that Chamber Parity is a critical factor influencing the operational behavior of the NDTech 2.0 system, particularly concerning valve torque measurements.

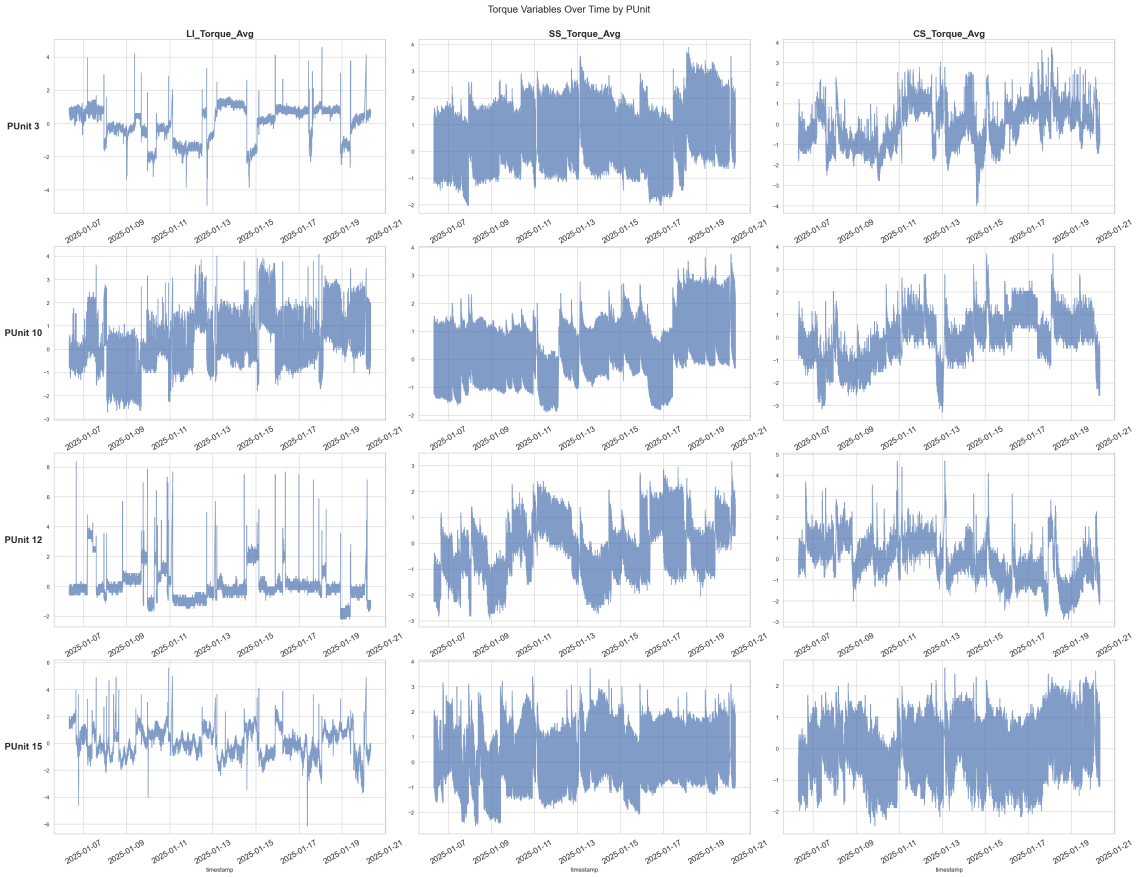


Figure 3.5: Torque Variables Over Time by PUnit.

Building upon the insights gained from the distribution analysis of torque variables, it becomes equally crucial to examine their behavior over time. Fig. 3.5 presents time-series

plots for LI_Torque_Avg, SS_Torque_Avg, and CS_Torque_Avg for each PUnit over the observed period in January 2025. This temporal perspective enables the identification of trends, cyclical patterns, and, most importantly, anomalous spikes or shifts that may indicate emerging issues or potential failures.

Across all PUnits, the torque variables generally exhibit a cyclical or oscillating pattern, reflecting the repetitive nature of the cork analysis process. However, closer inspection reveals variations and anomalies. For instance, in PUnit 3, the LI_Torque_Avg (top-left subplot) displays periods of relatively stable operation interspersed with pronounced, sharp increases or spikes in torque, particularly around January 10th and later. These sudden, high-magnitude deviations from the typical operating range are strong candidates for anomalous behavior, potentially indicating increased resistance or friction in the Load Inject valve mechanism, which could lead to wear or impending failure. Similar, though sometimes less dramatic, spikes can be observed for SS_Torque_Avg and CS_Torque_Avg across various PUnits.

Furthermore, subtle shifts in the baseline or the amplitude of the oscillations are also evident and warrant future investigation. For example, in PUnit 12, the LI_Torque_Avg seems to undergo a slight but sustained increase in its baseline torque values towards the latter half of the depicted period (e.g., after January 15th). Such gradual drifts, while not as immediately alarming as sharp spikes, can signify progressive degradation, component wear, or a change in system calibration that, if left unaddressed, could lead to performance degradation or eventual malfunction. These sustained changes, even if small in magnitude, are critical to monitor as they represent a different class of anomaly compared to instantaneous peaks.

The consistency of the overall operational envelopes for each torque variable across the different PUnits suggests a generally robust system. However, the intermittent spikes and subtle shifts observed in these time-series plots are crucial for future PdM efforts. Identifying the root causes of these deviations, whether they are transient operational variations, sensor noise, or genuine indicators of mechanical stress or degradation, will be a primary focus. The unsupervised learning models to be developed will aim to automatically detect these kinds of temporal anomalies, allowing for early intervention and a reduction in unscheduled downtime, thereby enhancing the overall reliability and efficiency of the NDTech 2.0 system.

Following the analysis of individual variable distributions and their temporal behavior, understanding the relationships between different operational parameters is crucial for a holistic view of the system's state. Fig. 3.6 presents correlation heatmaps for key torque and flow-related variables, segmented by each PUnit. These heatmaps illustrate the Pearson correlation coefficient between pairs of features, providing insights into how movements and gas flows within the system might influence each other. Positive correlations (represented by greener shades) indicate that variables tend to increase or decrease together, while negative correlations (browner shades) suggest an inverse relationship.

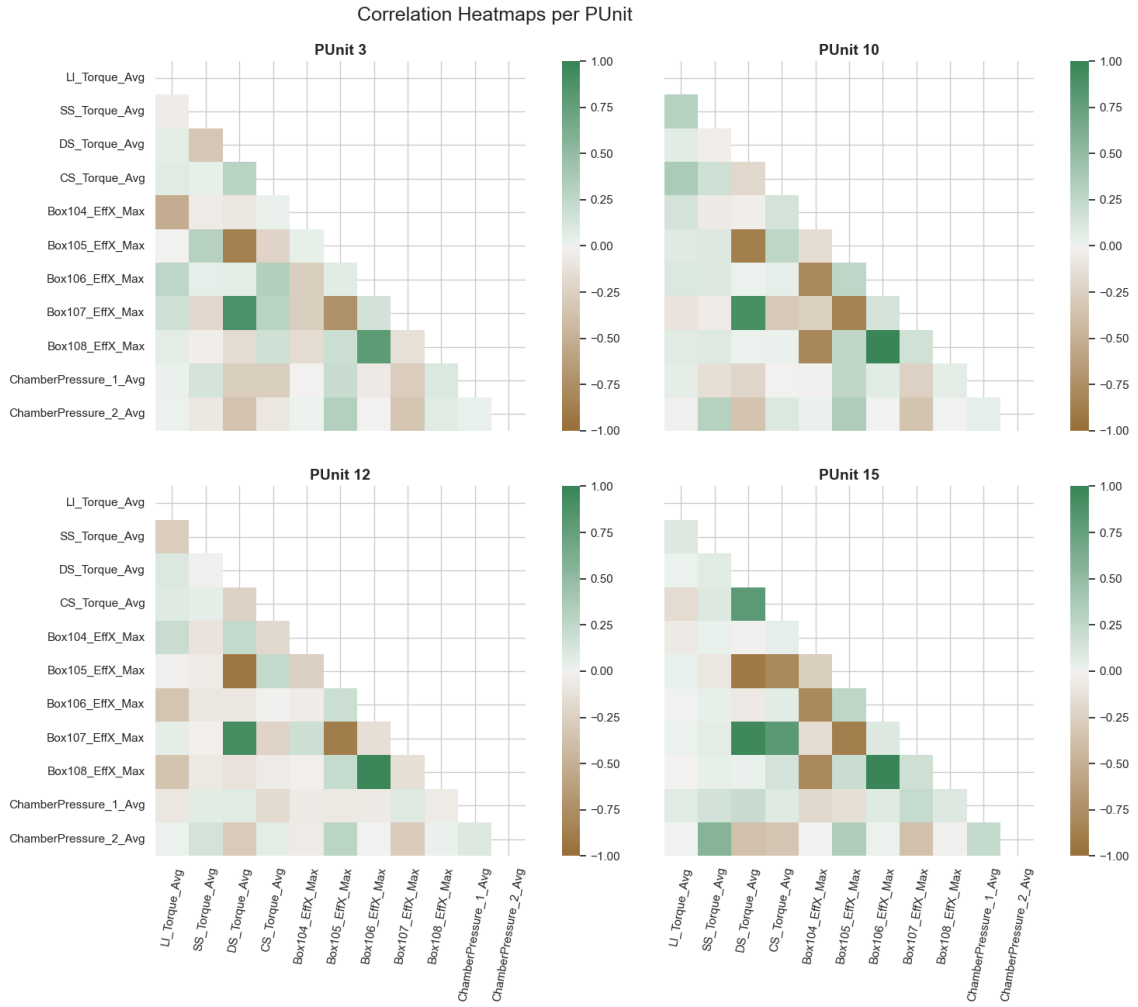


Figure 3.6: Correlation Heatmaps per PUnit.

A prominent observation across all PUnits is the generally weak or negligible correlation among the average torque variables (`LI_Torque_Avg`, `SS_Torque_Avg`, `DS_Torque_Avg`, `CS_Torque_Avg`) themselves. This suggests that the operational forces or resistance experienced by one valve's mechanism are largely independent of the others. This relative independence could simplify fault isolation, as a problem with one valve's torque is less likely to directly propagate to another.

However, more interesting patterns emerge when considering the relationships between torque variables and the Box variables (representing gas flow characteristics) or ChamberPressure variables. For instance, in PUnit 3, we observe a moderate positive correlation between `LI_Torque_Avg` and `Box104_EffX_Max`, and a moderate negative correlation with `Box104_OutY_Max`. This could indicate that specific load injector valve operations are directly influencing the gas flow dynamics through Box104, where higher torque might be associated with certain flow characteristics. Similar cross-correlations, though varying in strength and sign, can be seen for other PUnits. For example, PUnit 10 also shows a notable positive correlation between `LI_Torque_Avg` and `Box104_EffX_Max`.

A particularly consistent pattern across all PUnits is the strong negative correlation between `Box105_EffX_Max` and `Box106_EffX_Max`, as well as `Box107_EffX_Max` and `Box108_EffX_Max`. This strong inverse relationship between consecutive Box pairs suggests a compensatory or sequential flow mechanism within the column selector system. As flow increases in one path, it tends to decrease in the other, which is consistent with a system designed to precisely direct and manage nitrogen purging and sample flow. This provides a crucial baseline for normal operational relationships. Any significant deviation from these strong negative correlations could be a critical indicator of a malfunction, such as a valve sticking, a blockage, or a sensor reading error within the gas line system.

Furthermore, correlations involving `ChamberPressure_1_Avg` and `ChamberPressure_2_Avg` with torque or flow variables show varying strengths across PUnits, indicating nuances in how chamber pressure might be indirectly affected by valve operations or gas flow, or vice-versa.

3.4. Methodology

The system architecture, illustrated in Fig. 3.7, is designed to address the key objectives of real-time automated AD and the provision of interpretable explanations for failures or rare events, as outlined in Section 1.1. This architecture is fundamentally decomposed into two interconnected layers: the Failure Prediction Layer and the Explanation Layer. This dual-layer approach is inspired by contemporary PdM methodologies that integrate robust AD with clear interpretability to enhance operational decision-making, specifically drawing inspiration from the works presented in [62, 64].

The first layer, the Failure Prediction Layer, is responsible for identifying deviations from normal system behavior. Drawing inspiration from unsupervised DL models used in similar PdM contexts. Specifically, we will investigate AEs. These models are designed to

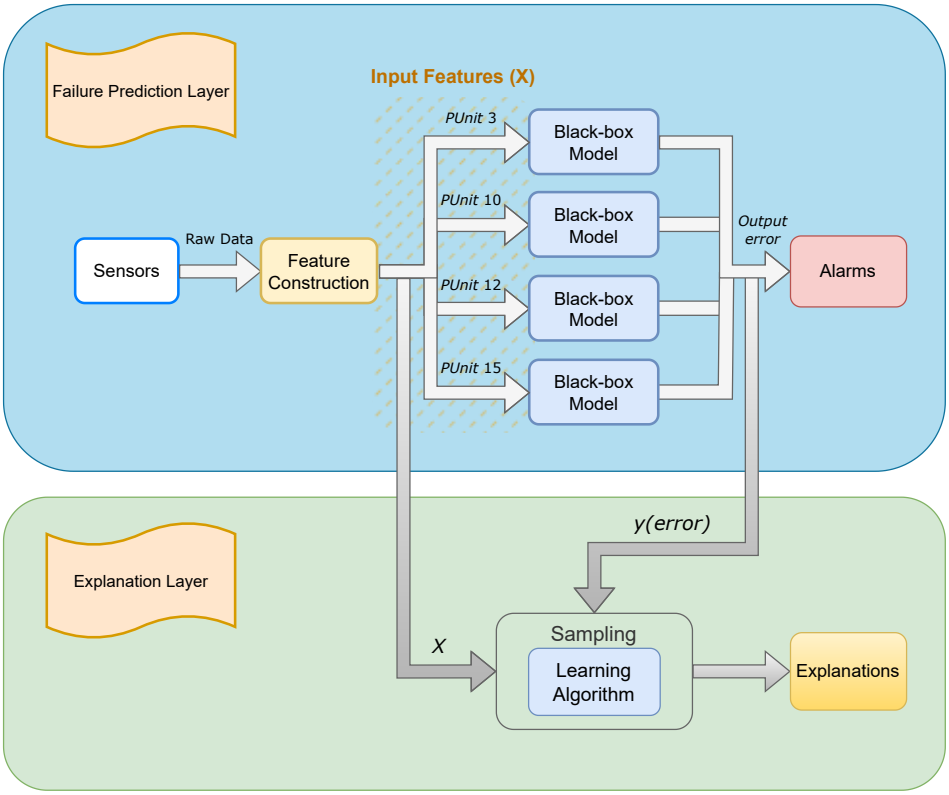


Figure 3.7: High-level view of the architecture.

learn the intricate patterns of normal operation from sensor data. When a new observation is processed, the model computes a metric (*Reconstruction Error* (RE)) to quantify the deviation from learned normal conditions. A significant deviation, exceeding a predefined threshold, triggers an alarm, signaling a potential anomaly or failure.

Running in parallel and online with the Failure Prediction Layer is the Explanation Layer. This layer's primary function is to provide human-understandable insights into why an anomaly or failure signal was triggered. While the detection layer identifies what is abnormal, the explanation layer aims to illuminate the causes of the anomaly. This is critical for operators, technicians, and managers to formulate effective repair plans and maintenance actions. This thesis will explore LIME framework* for this layer. The Explanation Layer receives the input features and the output from the Failure Prediction Layer. It then learns an interpretable mapping that describes the relationship between the input features and the anomaly signal. When an alarm is triggered, the Explanation Layer leverages this learned mapping to identify which specific sensors or input features contributed most significantly to the anomaly, thereby allowing for the identification of the component involved

*https://github.com/marcotcr/lime

in the failure. This approach facilitates explanations for particular detected failures, enhancing trust and usability for industry professionals.

By integrating these two layers, the proposed system aims to not only accurately detect potential failures in real-time but also to provide actionable, interpretable explanations, thereby significantly improving the efficiency and effectiveness of PdM operations on the NDTech system.

3.4.1 Failure Prediction Layer

The Failure Prediction Layer constitutes the foundational component of the real-time AD system. Its primary responsibility is to continuously monitor the processed operational data from the NDTech system and precisely identify any significant deviations from its established normal behavior. This layer operates entirely in an unsupervised learning paradigm, meaning it autonomously discerns and models the intricate patterns inherent to normal system operation without the need for pre-labeled anomaly data.

The input to this layer originates from the Feature Construction phase, where sensor readings undergo a critical transformation into a set of relevant, engineered features, denoted as X. A pivotal aspect of this data preparation is the partitioning of the comprehensive data stream into distinct subsets corresponding to each of the four functional PUnits (3, 10, 12, and 15) of the NDTech system. This disaggregation is indispensable, as the data is initially aggregated, and the operational characteristics and potential failure modes are unique to each PUnit. Consequently, individual AD models are trained and deployed for each PUnit, allowing for highly specialized and sensitive AD tailored to their specific dynamics.

Within this layer, we systematically implement and rigorously the performance of one prominent family of DL models: AEs. Each model type offers distinct advantages for AD in complex industrial environments—the detailed conceptual architectures can be found in Appendix A.1.

3.4.2 Explanation Layer

The Explanation Layer operates concurrently and in an online fashion with the Failure Prediction Layer, serving as a critical bridge between raw anomaly alarms and actionable intelligence. Its fundamental purpose is to demystify ”why” a particular failure signal was triggered by the detection models, thereby providing human-understandable insights.

While the Failure Prediction Layer excels at identifying ”what” is abnormal, the Explanation Layer is designed to illuminate the underlying causes of these detected anomalies. This capability is paramount for NDTech operators, technicians, and managers, enabling them to formulate precise repair plans, execute targeted maintenance actions, and generally enhance their decision-making processes.

This layer receives two primary inputs: the original input features X that were fed into the detection models at the time of the anomaly, and the anomaly score y (e.g., RE) generated by the Failure Prediction Layer. Using these inputs, the Explanation Layer constructs an interpretable mapping that articulates the relationship between the input features and the anomaly signal. When an alarm is activated, this learned mapping is leveraged to pinpoint which specific sensors or input features contributed most significantly to the anomaly, thereby facilitating the identification of the affected component within the NDTech system. Therefore, we will explore one methodology within this layer: LIME—in Appendix A.2 is explained how it works.

The integrated architecture, combining robust detection with interpretable explanations, aims to significantly improve the efficiency and effectiveness of PdM operations on the NDTech system.

3.4.3 Data Preparation and Chunking Strategy

The raw sensor data from the NDTech 2.0 system undergoes a crucial Feature Construction phase, transforming it into a structured input dataset, denoted as X. For DL models to effectively process this continuous time-series data, its organization into sequential segments is paramount. This is achieved through a precise chunking strategy, where the operational data for each PUnit is partitioned into overlapping sequences, or ”chunks”, using a sliding window approach. This method is fundamental because DL models, particularly LSTMs, necessitate sequential input to capture temporal dependencies and understand the evolving state of the PUnits. Furthermore, processing continuous, arbitrarily long time series is computationally inefficient; chunking mitigates this by creating fixed-size segments, facilitating both efficient training and real-time inference. The sliding window with overlap also closely simulates the online data processing environment, where new data continuously arrives, and predictions are made over recent historical windows, while also augmenting the training dataset for more robust feature learning by exposing the model to varied temporal contexts.

In this implementation, the data is segmented based on operational cycles. Each cycle comprises 8 timesteps. A value of 5 cycles defines the chunk size as 40 timesteps, meaning each input sequence to the model covers a 40-timestep temporal window, corresponding a 20-minute duration. The stride per chunk of 1 cycle, translating to a stride size of 8 timesteps, defines the overlap between consecutive chunks. This configuration ensures that each new chunk incorporates updated information while retaining significant temporal context from the previous segment, enabling continuous monitoring. The data pipeline sorts the data frame by timestamp, converting continuous and chamber-specific features into numerical arrays. Crucially, timestamps corresponding to the start and end of each chunk are also extracted and preserved, which is vital for later analysis and contextualizing explanations. The sliding window approach is utilized to efficiently create these overlapping chunks for both continuous and chamber variables, maintaining temporal integrity. The resulting chunks are then separately stored for each individual PUnit, a critical methodological choice to account for the unique operational characteristics and baseline behaviors inherent to each machine within the NDTech 2.0 system.

3.4.4 Training Process of the Failure Prediction Models

The AD models *Long Short-Term Memory Autoencoder* (LSTM-AE) and WAE-GAN with TCN are trained in an entirely unsupervised manner. This approach is necessitated by the inherent scarcity and often undefined nature of labeled anomaly data in real-world industrial settings. The core training objective for these models is to accurately reconstruct, or faithfully represent, the ”normal” operational behavior of each PUnit. This learned normality then serves as the fundamental baseline against which all subsequent operational data streams are continuously compared, allowing for the precise identification of any significant deviations that signal potential anomalies.

3.4.4.1 Offline Training

For each individual PUnit, a dedicated instance of each AD algorithm is trained. As referred before, this training phase is conducted offline, utilizing a substantial dataset comprising 10 days of historical operational data that are rigorously confirmed to correspond exclusively to the normal functioning of the machines. This duration aligns with established practices in PdM studies, such as [63, 64], providing sufficient data to capture the intricate normal patterns and inherent variability of the system. The rationale for using

only normal data is to teach the models to recognize deviations from this baseline, as any significant divergence would then be flagged as an anomaly.

Before training, an essential data preprocessing step was performed to handle potential extreme values in the training set. Specifically, for each continuous feature, the z-score was computed, and any observation exceeding a threshold of $|z| \geq 3$ was flagged as a potential outlier. These values were temporarily masked and subsequently imputed using a feature-wise RF Regressor. This approach models each affected feature based on the remaining features, enabling robust and non-linear imputation that respects the underlying multivariate structure of the data.

Imputation were performed separately for each PUnit, and the following number of anomalous rows were identified and corrected: 4415 rows for PUnit 3, 2668 for PUnit 10, 2690 for PUnit 12, and 6042 for PUnit 15. These values were then replaced by the respective RF predictions to ensure continuity and uniformity in the dataset. In Fig. 3.8 it can seen the trade-of between total and imputed records per PUnit.

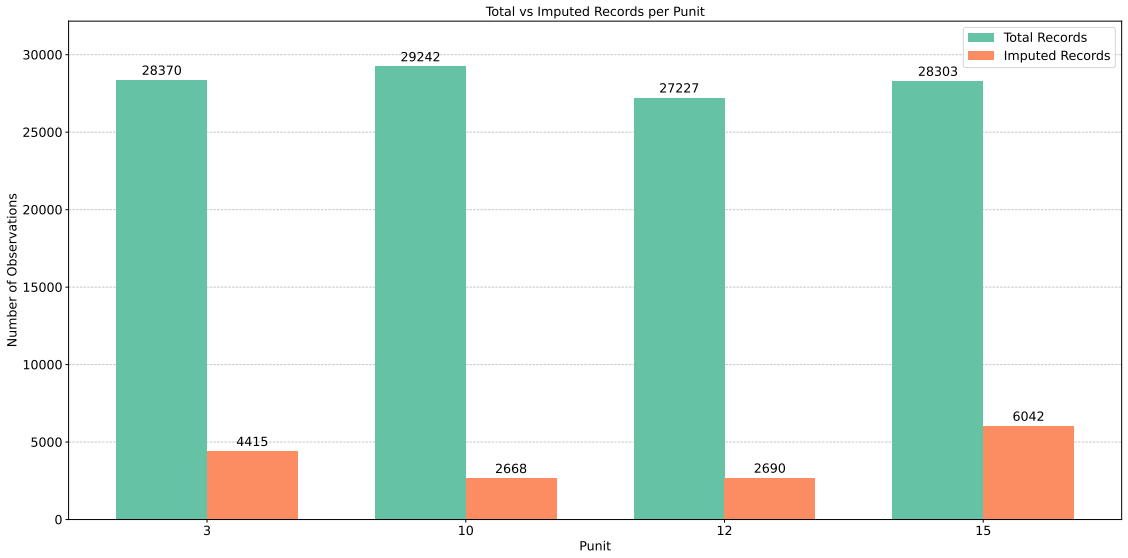


Figure 3.8: Total vs Imputed Records per PUnit.

After imputation, each dataset was standardized using a *StandardScaler*, which transforms each continuous variable to have zero mean and PUnit variance, improving the convergence behavior of the neural network models. Categorical variables (specifically, `Chamber`) were one-hot encoded to ensure compatibility with the model input layers. The `Punit` identifier, being already used to segment and train per-machine models, was dropped from the feature set to avoid redundancy.

During model training, the objective is to minimize a reconstruction loss. For AE, this is typically the MSE, while the WAE-GAN incorporates an additional adversarial loss to regularize the latent space and improve the representativeness of the generated reconstructions. The training process follows standard DL practices, including the use of the Adam optimizer and careful tuning of hyperparameters such as learning rate, batch size, and number of epochs for each model architecture and PUnit to ensure optimal learning of normal patterns. Training a separate model per PUnit accounts for individual machine characteristics and operational variability.

3.4.4.2 Training Error and Threshold Calculation

Upon completion of the offline training, the reconstruction errors for all normal training data chunks are computed. This distribution of normal reconstruction errors establishes the baseline for AD. A pivotal step is to establish a robust threshold to differentiate between normal and anomalous observations. A commonly adopted method, effective for the often skewed error distributions in AD, is based on statistical quartiles:

$$\text{Threshold} = Q3 + 3 \times (Q3 - Q1)$$

Here, $Q1$ represents the first quartile (25th percentile) and $Q3$ represents the third quartile (75th percentile) of the training reconstruction errors. The term $(Q3 - Q1)$ is the *Interquartile Range* (IQR). This formula, a variant of Tukey's fences [80], defines an upper boundary that robustly captures the vast majority of normal variations. The multiplier "3" for the IQR is a heuristic, chosen to be robust to outliers in the normal error distribution, providing a practical balance between sensitivity to true anomalies and minimizing false alarm rates, which are critical considerations in industrial applications.

3.4.5 Test Process

After model training and threshold establishment, the system transitions to the test phase, mimicking real-time operational monitoring to detect anomalies in unseen data.

3.4.5.1 Online Anomaly Detection

In the test process, the trained algorithms are applied to new (unseen operational) data, which is continuously processed into chunks using the identical "chunking strategy" employed during training. For each incoming chunk, the model computes its reconstruction error. This error is then compared against the pre-determined threshold derived from the

normal training data. If the test error for a given chunk exceeds the calculated threshold, it is flagged as a potential anomaly, producing a binary output (1 for anomalous, 0 for normal) for each processed chunk.

3.4.5.2 Low-Pass Filter Application

Raw anomaly scores can often be noisy, leading to transient spikes that do not represent true system failures. To mitigate this and increase the robustness of the solution, a low-pass filter is applied to the binary anomaly output stream. This filter, often implemented as a simple moving average or exponential smoothing, effectively smooths out short-term fluctuations and highlights longer-term trends in the anomaly signal. There is a parameter alpha that controls the degree of smoothing; higher alpha values result in less smoothing (more sensitive to rapid changes), while lower alpha values result in more smoothing (less sensitive, requiring a more sustained anomaly to trigger an alert). This post-processing step significantly reduces false positives and ensures that only persistent and significant deviations trigger an alarm for the Explanation Layer, enhancing the system's practical utility.

3.4.6 Evaluation Metrics

To rigorously assess the performance of the AD models, a clear evaluation approach is established based on the temporal overlap between predicted anomaly intervals and actual failures reported by maintenance teams. This approach ensures that evaluation directly reflects real-world operational relevance.

Specifically, the following definitions are applied for classification:

- **True Positive (TP):** An output anomaly interval is considered a true positive if it significantly overlaps with a failure event reported by maintenance teams. This indicates a successful detection of a genuine fault.
- **False Positive (FP):** An output anomaly interval is classified as a false positive (false alarm) if it does not overlap with any failure reported by maintenance. These represent unnecessary alerts, which can lead to wasted resources and operator fatigue.
- **False Negative (FN):** If a failure is reported by maintenance but does not correspond to any output anomaly interval from the model, this event is considered a

false negative (a missed alarm). Missed detections are critical as they can lead to unexpected downtime, increased repair costs, and safety issues.

Based on these definitions, the model's performance is quantified using key metrics that are crucial in PdM contexts, aligning with practices in [64].

We report our primary results using the **F1-score**, which is the harmonic mean of Precision and Recall. Precision, calculated as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$, measures the proportion of correctly identified anomalies among all predicted anomalies, reflecting the model's accuracy in avoiding false alarms. Recall, calculated as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$, measures the proportion of actual anomalies that were successfully detected, reflecting the model's ability to minimize missed failures. The F1-score provides a balanced assessment, particularly valuable in AD where class imbalance is common, as it penalizes models that perform poorly on either precision or recall.

In addition to individual performance metrics, we analyze the **confusion matrix** for each model. This provides a comprehensive view of classification outcomes, helping identify tendencies such as over-sensitivity or under-detection. The confusion matrix includes all three components (TP, FP, FN), allowing for a more granular evaluation of model behavior across various conditions.

In cases where two or more models achieve equivalent performance under the primary evaluation, the **F1-score** is used as a tiebreaker. The model with the highest F1-score among the tied candidates is selected as the better option, given its balanced emphasis on both false alarms and missed detections.

These metrics collectively provide a robust and interpretable basis for assessing the practical utility and effectiveness of the AD models within the NDTech 2.0 system.

3.4.7 Integration with Explanation Layer

The final crucial step in the proposed system is the seamless integration of the Failure Prediction Layer with the Explanation Layer. Once an anomaly is detected and confirmed (post-low-pass filter and thresholding), the system triggers the explanation process, bridging the gap between detection and actionable insight.

When an alarm is activated for a specific PUnit, the raw input features X that led to this anomaly, along with the calculated anomaly score y (reconstruction/output error), are passed to the Explanation Layer. This layer then leverages LIME to provide interpretable

insights into the detected anomaly. This multi-pronged approach offers versatility in diagnostic reporting. LIME creates a local, interpretable surrogate model around the anomalous instance. By perturbing the input features and observing the black-box model's response, LIME identifies the most influential features responsible for the anomaly. This provides a localized understanding of the anomaly's root causes. This comprehensive integration ensures the system not only alerts operators to anomalies but also empowers them with critical diagnostic information, significantly reducing downtime and improving maintenance efficiency by directing attention to the precise cause of the deviation, aligning with the architectural vision for NDTech 2.0's AD and explanation capabilities presented in Fig. 3.7.

4. Experimental Study

This chapter details the implementation of the failure prediction models, including their architecture, training methodology, and the results obtained, related to the failure prediction layer. It also covers the application of interpretability technique to provide actionable insights into detected failures, related to the explanation layer.

4.1. Implementation of Anomaly Detection Models

This section describes the practical implementation of the AD models, specifically focusing on the LSTM-AE and WAE-GAN. Both models process continuous features (e.g., torque, position, temperature) and categorical chamber identifiers. The categorical features are transformed into dense vectors via an embedding layer and then concatenated with continuous features to form a unified input. The hyperparameter tuning for each model can be found in Appendix A.3, where each table includes the identifiers of the trained models, which are referenced in the subsequent analysis.

4.1.1 LSTM Autoencoder

The LSTM-AE is fundamentally structured upon a classic encoder-decoder paradigm, designed to learn a compressed representation of normal operational patterns and reconstruct the original input sequence. This architecture is conceptually illustrated in Fig. 4.1.

The encoder component is a multi-layered LSTM network that distills input sequences into a compact latent representation. The decoder, conversely, reconstructs the original input sequence from this latent vector, utilizing bidirectional LSTM layers for superior reconstruction quality. The output layer is designed to reconstruct both the continuous features and the chamber embeddings.

The training objective for the LSTM-AE is to minimize the discrepancy between original and reconstructed sequences using a hybrid loss function. This function combines MSE for continuous features and Cross-Entropy Loss for categorical chamber features. Model optimization was conducted using the Adam optimizer with an adaptive learning rate scheduler, and gradient clipping was applied to mitigate exploding gradients. Each model was trained for 100-150 epochs, with losses meticulously logged to monitor convergence. The AD threshold was determined based on the statistical distribution of REs

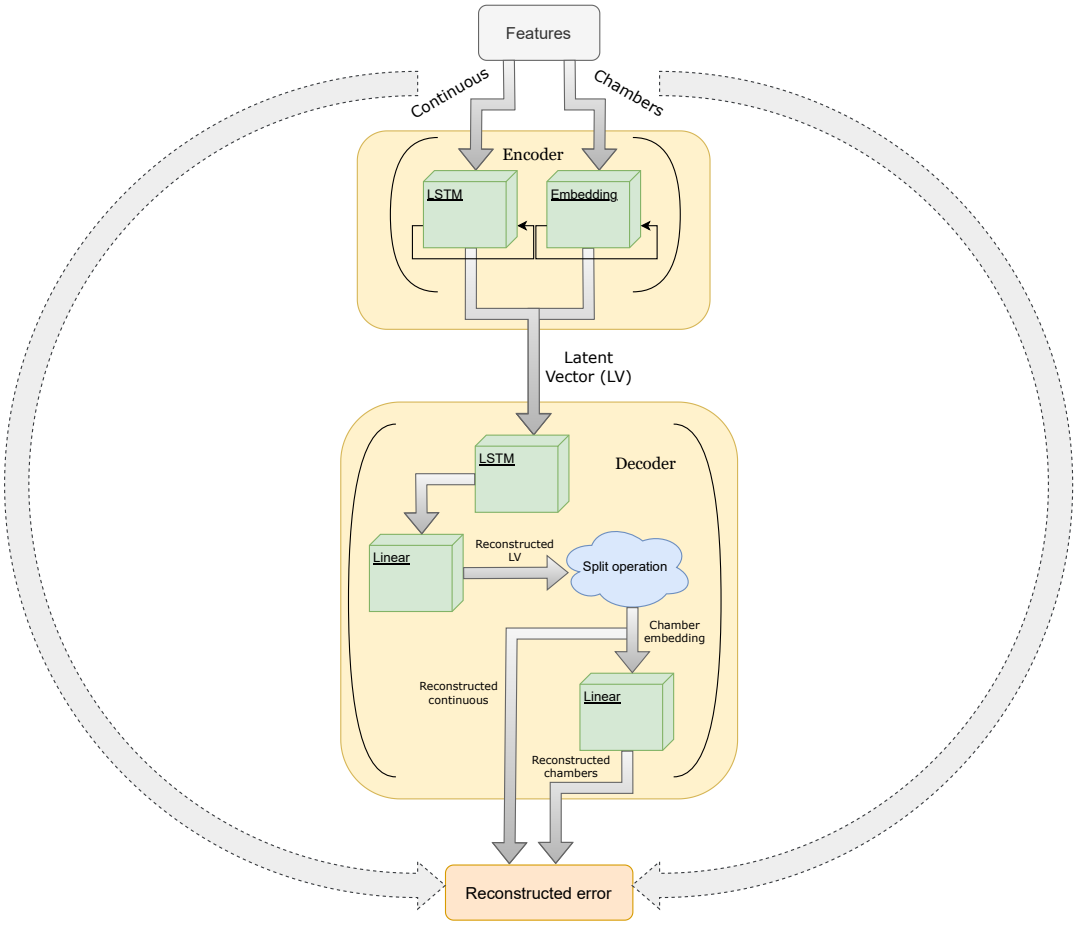


Figure 4.1: Architectural Diagram of the LSTM-AE. The encoder maps input sequences to a latent vector, which the decoder subsequently uses to reconstruct the original input, thereby learning a compressed representation of normal operational patterns.

from normal training data. Further details on the model architecture, training process, and feature definitions are provided in Appendix B.1.

4.1.2 Wasserstein Autoencoder Generative Adversarial Network

The WAE-GAN architecture is composed of three synergistic components: an Encoder, a Decoder, and a Discriminator. Both the Encoder and Decoder leverage the power of TCNs, known for their efficacy in processing sequential data, while the Discriminator is an LSTM network adept at discerning temporal patterns. The architectural blueprint is conceptually depicted in Fig. 4.2.

The Encoder maps high-dimensional input data into a lower-dimensional latent space. The Decoder reconstructs the original input sequence from this latent vector. The LSTM-Discriminator operates within the latent space, distinguishing between latent vectors from

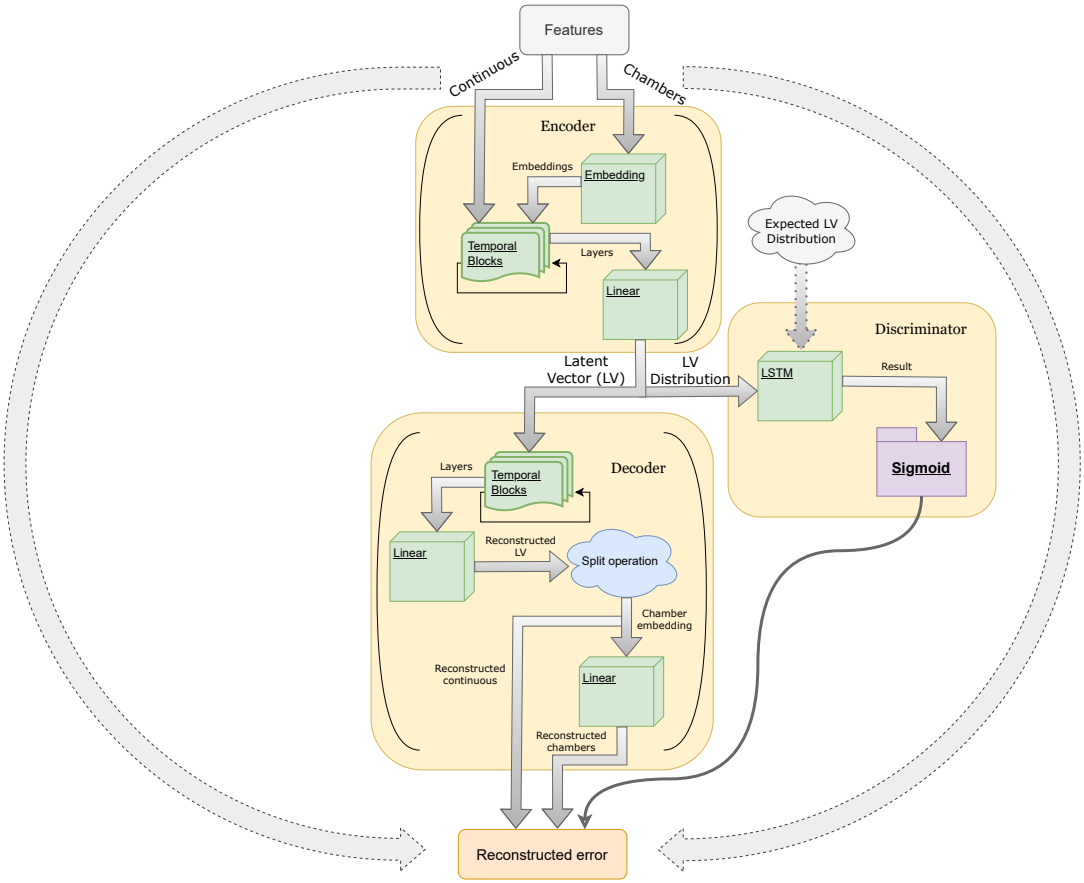


Figure 4.2: Architectural Diagram of the WAE-GAN. The Encoder maps input sequences to a latent representation, which the Decoder subsequently uses for reconstruction. The Discriminator distinguishes between latent vectors sampled from a prior distribution (Gaussian) and those generated by the Encoder.

the Encoder's output and those sampled from a standard multivariate Gaussian distribution.

The training of WAE-GAN models is a sophisticated multi-objective optimization problem. It involves iteratively training the Discriminator and the combined Encoder-Decoder (as the autoencoder/generator) in an asymmetric fashion, with the Encoder and Decoder updated twice for every single Discriminator update. This ratio was empirically determined to maintain a balanced adversarial game and promote stable convergence. The Encoder-Decoder is trained to minimize RE (using MSE for continuous features and Cross-Entropy for categorical features) and align its latent space with a prior distribution by "fooling" the Discriminator. The Discriminator is trained to distinguish between real and generated latent samples. Gradient clipping and separate Adam optimizers are used for training

stability. Further comprehensive details on the TCN Temporal Block, specific layer configurations, loss function derivations, and detailed training process for WAE-GAN are provided in Appendix B.2.

4.2. Failure Prediction Results

This section presents a comprehensive analysis of the failure prediction performance achieved by the trained LSTM-AE and WAE-GAN models for each PUnit. The evaluation adheres to the rigorous metrics and definitions established in Section 3.4.6, focusing on the F1-score, Precision, and Recall to quantify the models' efficacy in identifying critical operational failures. For each PUnit, the best-performing LSTM-AE and WAE-GAN model, selected based on a holistic assessment of its training convergence, validation performance, and overall performance metrics.

To ensure comprehensive analysis while maintaining conciseness, this section focuses on presenting the primary performance metrics and failure prediction plots for the selected best model(s) for each PUnit. Detailed visual insights, including reconstruction loss progressions, RE distributions, statistical summaries, and confusion matrices for all evaluated models, can be found in Appendices C.1 and C.2.

4.2.1 LSTM-AE Performance

The training process for each PUnit involved evaluating multiple LSTM-AE architectures and configurations. The selection of the best model for each PUnit was a meticulous process, primarily guided by the reconstruction loss progression during training, the distribution of REs on both training and test datasets, and critically, the resulting classification performance metrics derived from the confusion matrices. Some models demonstrating stable convergence, minimal overfitting, and superior F1-scores on the unseen test data were prioritized.

PUnit 3 Model `lstm-ae-punit3-2` was the best performer among three candidates, showing strong F1-score and Recall (Table 4.1). Fig. 4.3 illustrates its anomaly scores over time. The model detected 3 severe and 2 non-severe failures but recorded one false positive (2025-01-09) and missed a failure (2025-01-19).

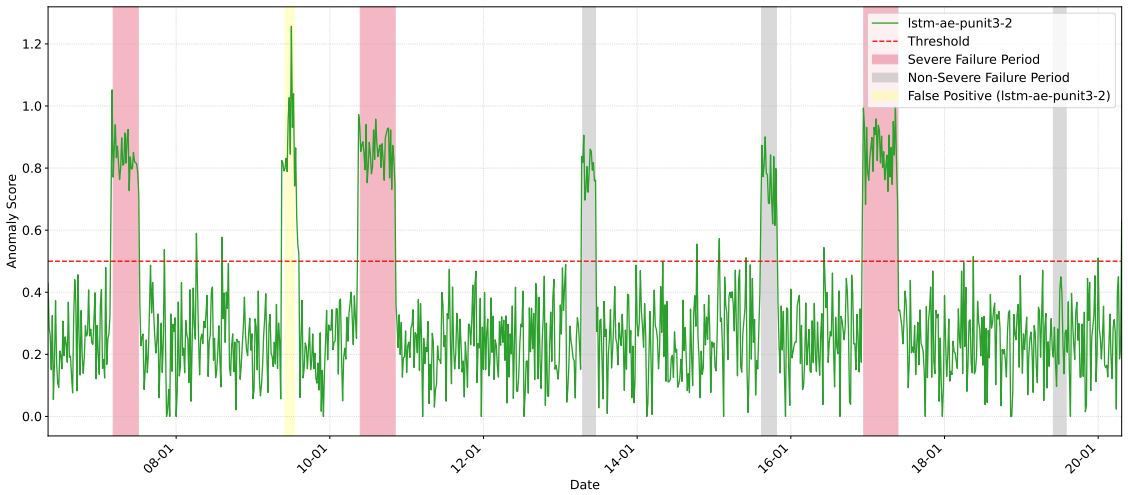


Figure 4.3: Failure Prediction for Model `lstm-ae-punit3-2` - PUnit 3.

Table 4.1: Performance Measures for All Trained LSTM-AE Models - PUnit 3

Model ID	Precision	Recall	F1-score
lstm-ae-punit3-1	1.00	0.50	0.67
lstm-ae-punit3-2	0.83	0.83	0.83
lstm-ae-punit3-3	1.00	0.67	0.80

PUnit 10 Model `lstm-ae-punit10-1` achieved the highest F1-score and Precision (Table 4.2). As shown in Fig. 4.4, it detected 3 severe and 1 non-severe failures but missed one failure (2025-01-18).

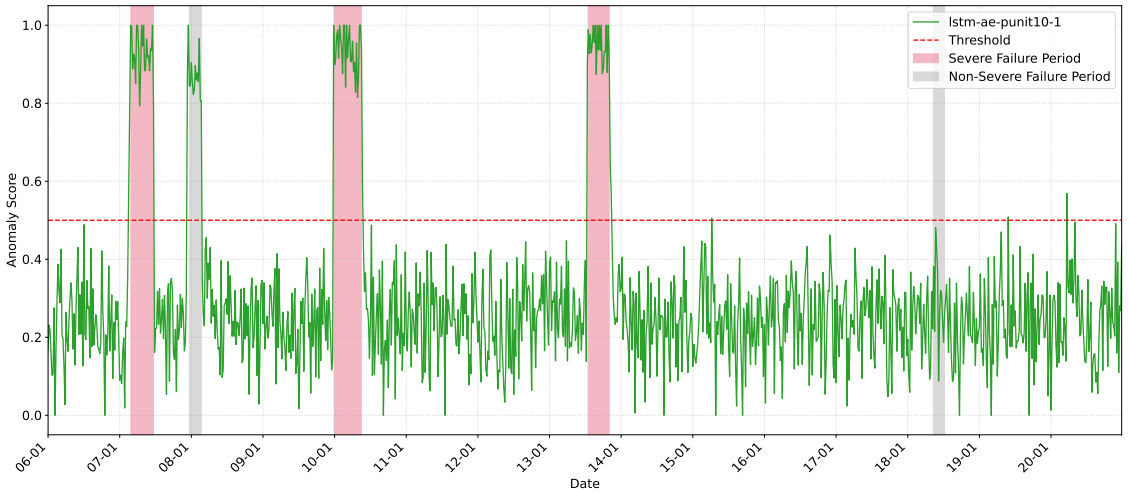


Figure 4.4: Failure Prediction for Model `lstm-ae-punit10-1` - PUnit 10.

PUnit 12 Model `lstm-ae-punit12-1` offered the best overall performance in terms of F1-score (Table 4.3). It successfully identified 1 severe and 2 non-severe failures (Fig. 4.5), but missed one failure (2025-01-07) and had a false positive (2025-01-15).

Table 4.2: Performance Measures for All Trained LSTM-AE Models - PUnit 10

Model ID	Precision	Recall	F1-score
lstm-ae-punit10-1	1.00	0.80	0.89
lstm-ae-punit10-2	0.80	0.80	0.80
lstm-ae-punit10-3	0.67	0.40	0.50

Figure 4.5: Failure Prediction for Model `lstm-ae-punit12-1` - PUnit 12.

Table 4.3: Performance Measures for All Trained LSTM-AE Models - PUnit 12

Model ID	Precision	Recall	F1-score
lstm-ae-punit12-1	0.75	0.75	0.75
lstm-ae-punit12-2	0.60	0.75	0.67
lstm-ae-punit12-3	1.00	0.50	0.67

PUnit 15 Models `lstm-ae-punit15-1` and `lstm-ae-punit15-2` had equal metrics (Precision, Recall, F1-score), suggesting modest performance (Table 4.4). Each detected 1 severe and 2 non-severe failures but exhibited two false positives and missed two failures, as shown in Fig. 4.6.

Table 4.4: Performance Measures for All Trained LSTM-AE Models - PUnit 15

Model ID	Precision	Recall	F1-score
lstm-ae-punit15-1	0.60	0.60	0.60
lstm-ae-punit15-2	0.60	0.60	0.60
lstm-ae-punit15-3	0.67	0.40	0.50

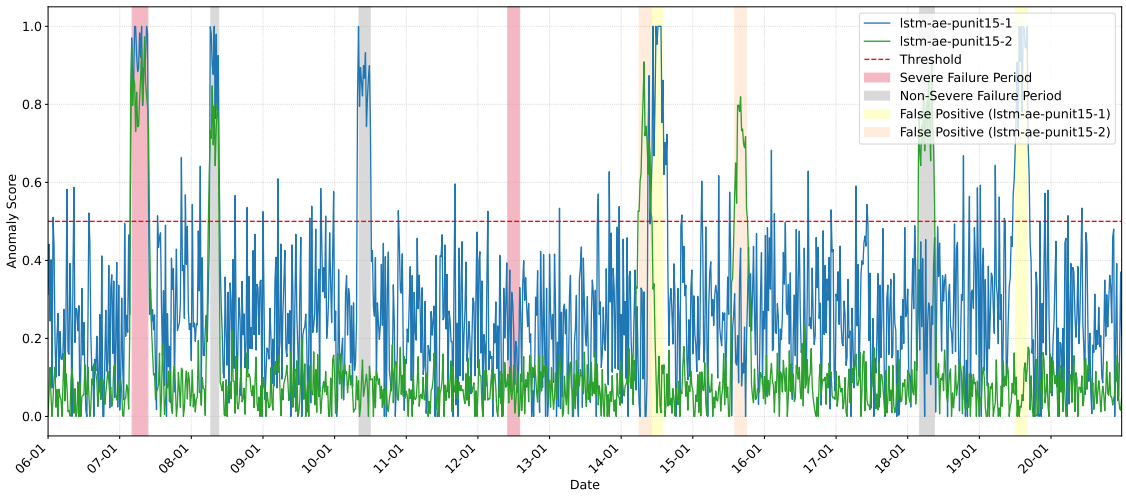


Figure 4.6: Failure Prediction for Models `lstm-ae-punit15-1` and `lstm-ae-punit15-2` - PUnit 15.

4.2.2 WAE-GAN Performance

This section presents the performance of the trained WAE-GAN models across all PUnits. For each PUnit, the analysis focuses on AD visualizations and comprehensive performance metrics for the optimal model. As with the LSTM-AE models, the optimal model is selected based on performance metrics on the test set.

PUnit 3 Out of all trained models, `wae-gan-punit3-3` achieved the highest performance for PUnit 3, with an F1-score of 0.92 (see Table 4.5). The model detected three severe and three non-severe failures, along with a single false positive on 2025-01-09 (Fig. 4.7).

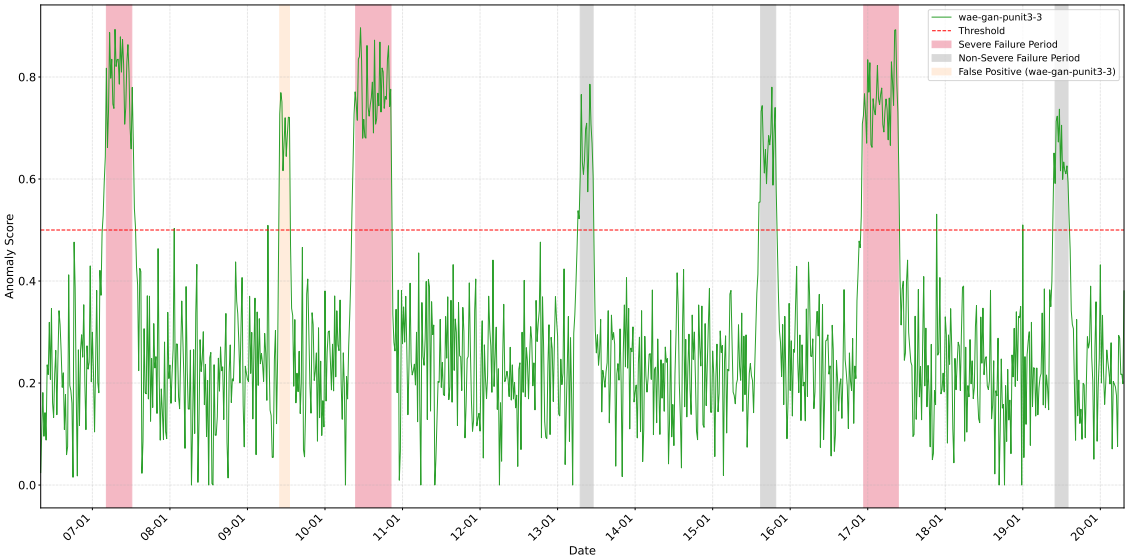


Figure 4.7: Failure Prediction for Model `wae-gan-punit3-3` - PUnit 3.

Table 4.5: Performance Measures for All Trained WAE-GAN Models - PUnit 3

Model ID	Precision	Recall	F1-score
wae-gan-punit3-1	0.83	0.83	0.83
wae-gan-punit3-2	0.83	0.83	0.83
wae-gan-punit3-3	0.86	1.00	0.92
wae-gan-punit3-4	0.80	0.67	0.73

PUnit 10 Model wae-gan-punit10-4 delivered the best results for PUnit 10 with an F1-score of 0.91 (Table 4.6). It successfully detected three severe and two non-severe failures, with one false positive occurring on 2025-01-17 (Fig. 4.8).

Figure 4.8: Failure Prediction for Model wae-gan-punit10-4 - PUnit 10.

Table 4.6: Performance Measures for All Trained WAE-GAN Models - PUnit 10

Model ID	Precision	Recall	F1-score
wae-gan-punit10-1	0.75	0.60	0.67
wae-gan-punit10-2	0.67	0.80	0.73
wae-gan-punit10-3	0.80	0.80	0.80
wae-gan-punit10-4	0.83	1.00	0.91

PUnit 12 The optimal model for PUnit 12, wae-gan-punit12-4, showed balanced and stable performance, achieving an F1-score of 0.89 (Table 4.7). It detected two severe and two non-severe failures, with one false positive on 2025-01-15 (Fig. 4.9).

Figure 4.9: Failure Prediction for Model `wae-gan-punit12-4` - PUnit 12.

Table 4.7: Performance Measures for All Trained WAE-GAN Models - PUnit 12

Model ID	Precision	Recall	F1-score
`wae-gan-punit12-1`	0.67	0.50	0.57
`wae-gan-punit12-2`	0.75	0.75	0.75
`wae-gan-punit12-3`	0.75	0.75	0.75
`wae-gan-punit12-4`	0.80	1.00	0.89

PUnit 15 Model `wae-gan-punit15-4` performed best for PUnit 15, achieving an F1-score of 0.91 (Table 4.8). It identified two severe and three non-severe failures, with one false positive on 2025-01-20 (Fig. 4.10).

Figure 4.10: Failure Prediction for Model `wae-gan-punit15-4` - PUnit 15.

Table 4.8: Performance Measures for All Trained WAE-GAN Models - PUnit 15

Model ID	Precision	Recall	F1-score
wae-gan-punit15-1	1.00	0.80	0.89
wae-gan-punit15-2	0.80	0.80	0.80
wae-gan-punit15-3	0.80	0.80	0.80
wae-gan-punit15-4	0.83	1.00	0.91

4.2.3 Summary of Performance

Across all evaluated PUnits, the WAE-GAN models consistently achieved notably higher F1-scores, demonstrating their superior capability for this application. This superiority is attributed to its adversarial training mechanism, which enables the generator to produce more realistic reconstructions and leads to more precise decision boundaries for AD. Furthermore, WAE-GAN models required significantly fewer epochs to converge (typically around 30, compared to 100–120 for LSTM-AE), resulting in faster training and deployment. They also exhibited greater stability during normal operating periods, contributing to reduced false positives and enhanced reliability. A key architectural strength of WAE-GAN is its enforcement of a Gaussian prior in the latent space, which facilitates clearer separation between normal and anomalous samples and aids robust threshold definition. Furthermore, visualizations of the latent spaces (PCA, t-SNE, and latent dimension histograms) of all WAE-GAN optimal models, which further support the distinct and structured latent representation achieved are provided in Appendix C.3.

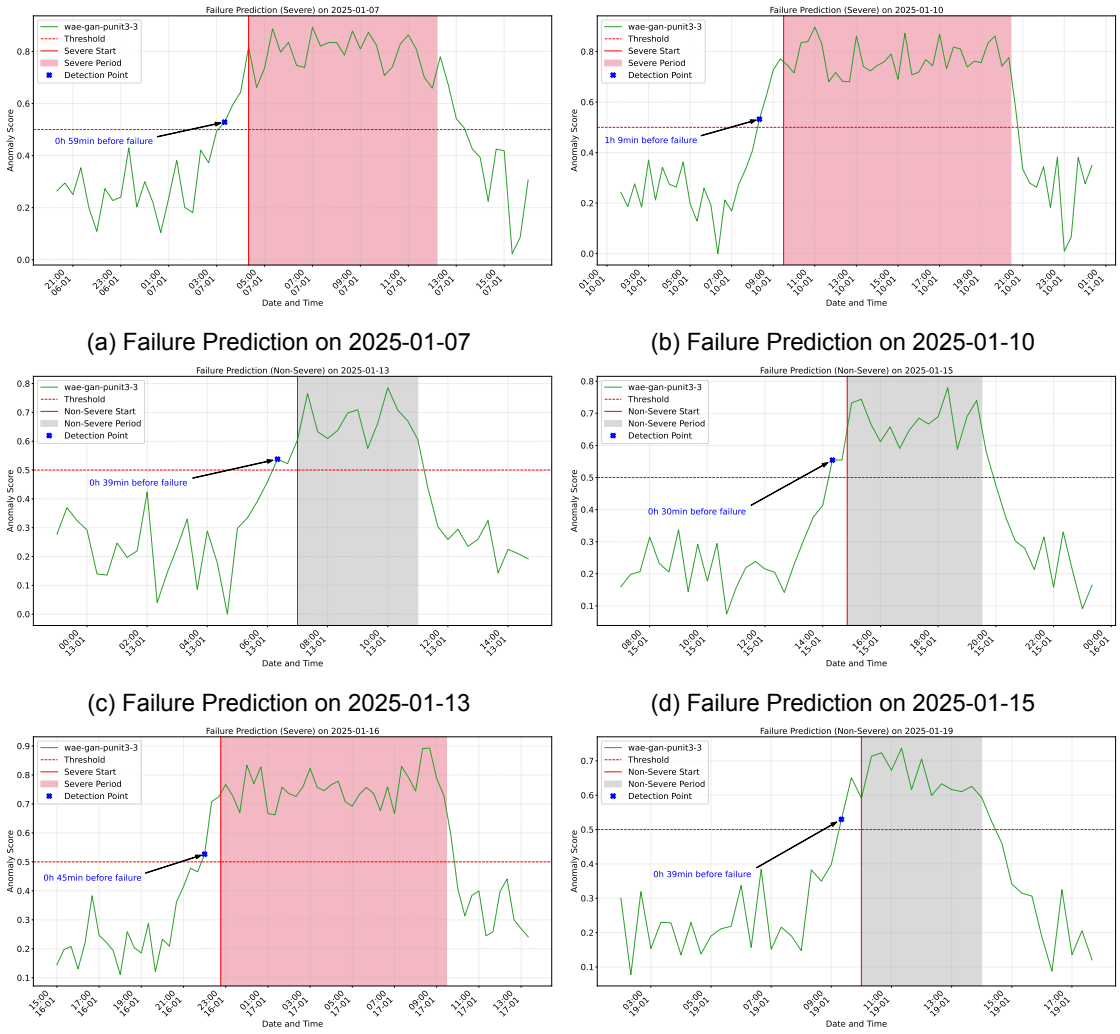
Table 4.9 summarizes the performance of the best WAE-GAN model for each PUnit, showcasing their F1-scores, Precision, and Recall.

Table 4.9: Summary of Best WAE-GAN Model Performance Measures per PUnit.

PUnit	Precision	Recall	F1-score
PUnit 3	0.86	1.00	0.92
PUnit 10	0.83	1.00	0.91
PUnit 12	0.80	1.00	0.89
PUnit 15	0.83	1.00	0.91

4.2.4 Lead Time Analysis

For practical deployment in PdM, early detection is crucial. The lead times of failure predictions across all PUnits ranged between approximately 30 to 70 minutes, enabling proactive interventions. Fig. 4.11-4.14 illustrate how the anomaly score clearly rises above the threshold preceding actual failure events for various failure events across different PUnits, demonstrating consistent and practical lead times across units, which underscores the WAE-GAN effectiveness.



(a) Failure Prediction on 2025-01-07

(b) Failure Prediction on 2025-01-10

(c) Failure Prediction on 2025-01-13

(d) Failure Prediction on 2025-01-15

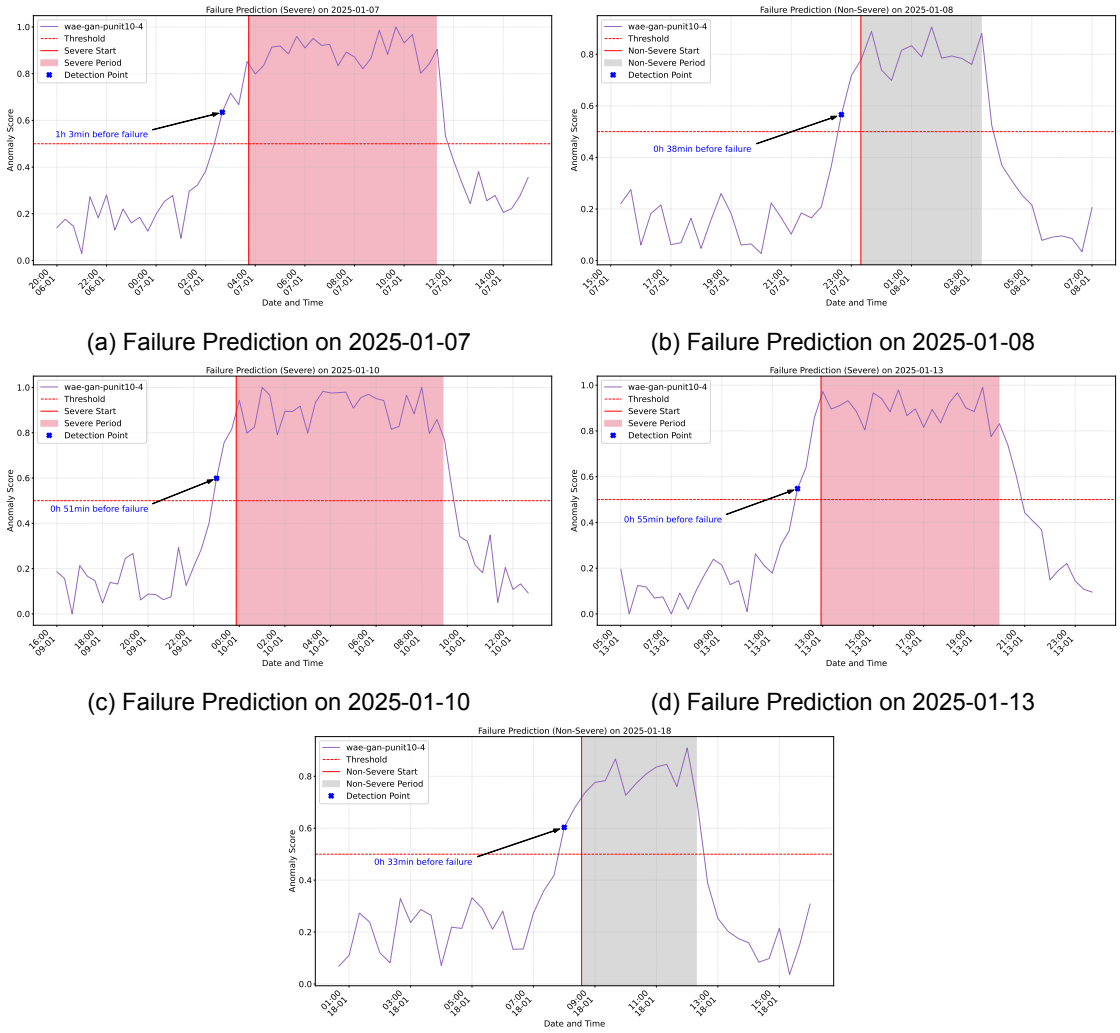
(e) Failure Prediction on 2025-01-16

(f) Failure Prediction on 2025-01-19

Figure 4.11: Lead Time of Failure Prediction for Model `wae-gan-punit3-3` - PUnit 3.

For PUnit 3, model `wae-gan-punit3-3` consistently provided actionable lead times, as shown in Fig. 4.11. A severe failure on 2025-01-10 was predicted 1 hour and 9 minutes in advance, while a non-severe event on 2025-01-13 showed a lead time of 39 minutes. Another severe failure on 2025-01-16 was detected 45 minutes prior, and a non-severe failure on 2025-01-19 was identified 39 minutes before occurrence.

For PUnit 10, model `wae-gan-punit10-4` demonstrated similarly strong predictive capabilities (Fig. 4.12). A severe failure on 2025-01-07 was detected 1 hour and 3 minutes in advance, and another on 2025-01-10 was predicted 51 minutes prior. Non-severe failures on 2025-01-08 and 2025-01-18 were detected with lead times of 38 and 33 minutes, respectively. These results indicate that severe failures generally exhibited longer lead times (51–63 minutes) compared to non-severe ones (33–40 minutes), suggesting different degradation profiles that can support prioritization of maintenance actions.



(a) Failure Prediction on 2025-01-07

(b) Failure Prediction on 2025-01-08

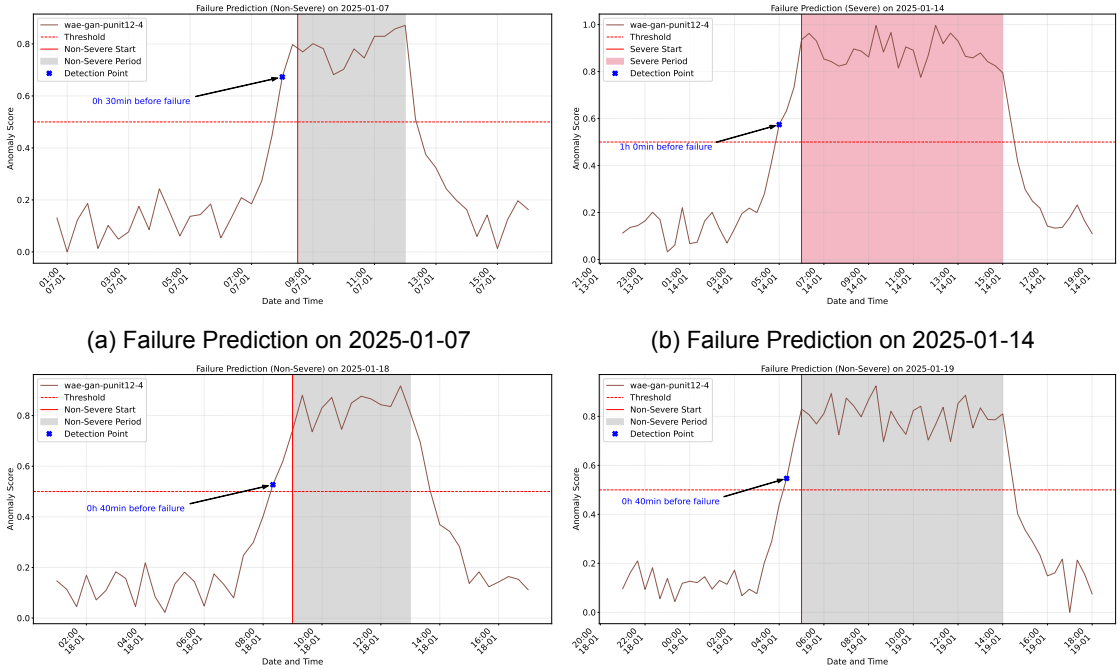
(c) Failure Prediction on 2025-01-10

(d) Failure Prediction on 2025-01-13

(e) Failure Prediction on 2025-01-18

Figure 4.12: Lead Time of Failure Prediction for Model `wae-gan-punit10-4` - PUnit 10.

For PUnit 12, model `wae-gan-punit12-4` also provided significant predictive value (Fig. 4.13), detecting a severe failure on 2025-01-14 with a 1-hour lead time, and non-severe failures on 2025-01-07, 2025-01-18, and 2025-01-19 with lead times of 30, 40, and 40 minutes, respectively.



(a) Failure Prediction on 2025-01-07

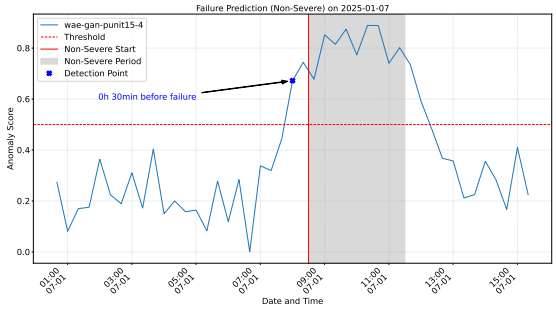
(b) Failure Prediction on 2025-01-14

(c) Failure Prediction on 2025-01-18

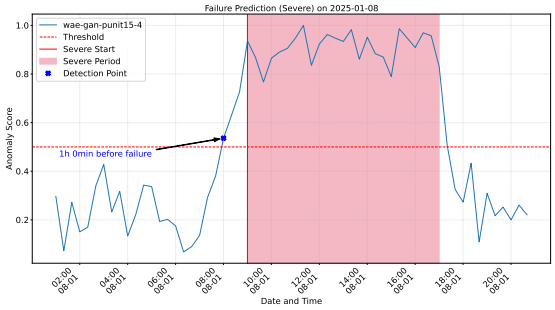
(d) Failure Prediction on 2025-01-19

Figure 4.13: Lead Time of Failure Prediction for Model `wae-gan-punit12-4` - PUnit 12.

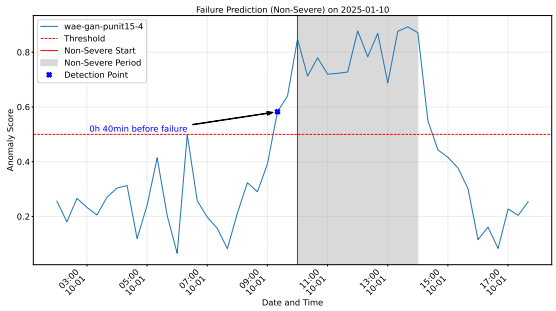
Lastly, model `wae-gan-punit15-4` for PUnit 15 consistently delivered substantial advance warnings (Fig. 4.14), predicting a severe failure on 2025-01-08 with a 1-hour lead time and another on 2025-01-12 with 40 minutes. Non-severe events on 2025-01-07, 2025-01-10, and 2025-01-18 were each detected with lead times ranging from 30 to 40 minutes. In all cases, anomaly scores showed a clear and sustained rise above the threshold preceding each event, reinforcing the reliability of the models' forecasts. These consistent and practical lead times across multiple units and failure severities underscore the WAE-GAN effectiveness in enabling timely and proactive maintenance decisions within a predictive maintenance context.



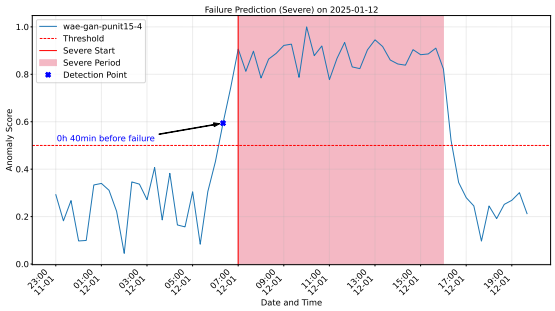
(a) Failure Prediction on 2025-01-07



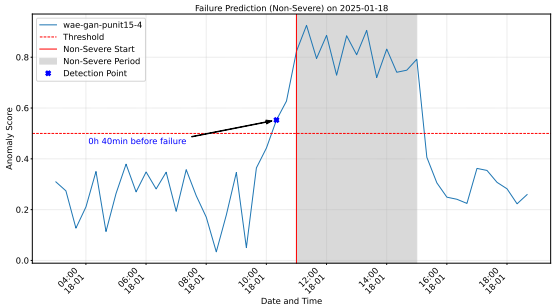
(b) Failure Prediction on 2025-01-08



(c) Failure Prediction on 2025-01-10



(d) Failure Prediction on 2025-01-12



(e) Failure Prediction on 2025-01-18

Figure 4.14: Lead Time of Failure Prediction for Model `wae-gan-punit15-4` - PUnit 15.

4.3. Failures Explanation Results

The ability to not only detect failures but also to understand their root causes is paramount for effective PdM in complex industrial systems like NDTech 2.0. This section presents the crucial results of applying the LIME framework to clarify the contributing factors behind detected severe failures. This analysis aims to provide actionable insights for maintenance teams, directing attention to the specific sensor readings that deviate from normal behavior and explaining ”why” a failure was classified.

LIME’s interpretability is based on identifying specific binary conditions or rules for features that locally explain a particular failure instance. These rules, presented as numerical intervals, indicate that if a feature’s value falls within the specified range, it was identified as a significant contributor to the anomaly. Features with positive contributions (visually represented in red in LIME explanations) increase the anomaly score, suggesting a

potential failure. Conversely, negative contributions (green) push the prediction towards normality. The following visualizations illustrate both the magnitude and direction of each feature's impact on the anomaly score for representative severe failure events. Comprehensive LIME explanations for all non-severe detected failures, and any other severe failures not explicitly detailed here, are provided in Appendix C.4.

4.3.1 PUnit 3

This section details the LIME explanations for severe failures detected by the optimal `wae-gan-punit3-3` model in PUnit 3. These explanations highlight how specific feature conditions contributed to high anomaly scores, providing both commonly identified patterns and instance-specific insights for maintenance personnel.

Failure on 2025-01-07 Fig. 4.15 prominently highlights that this instance is primarily explained by positive contributions from `CS_Torque_Max` being in a specific low negative range, alongside distinct conditions in `Box106_OutY_Avg`, `Box108_EffX_Min`, and `Box104_OutY_Max`. Conversely, `Box107_OutY_Max` and `ChamberPressure_1_Avg` show negative contributions, indicating their values were aligned with normal operation for this specific failure. The detailed feature rules that contributed to this failure and their contributions are summarized in Table 4.10.

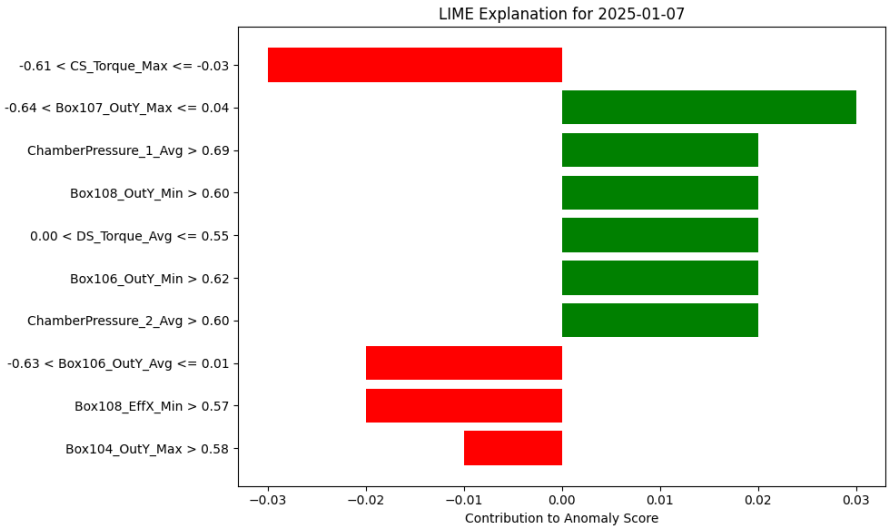


Figure 4.15: LIME explanation for severe failure on 2025-01-07 detected by `wae-gan-punit3-3` in PUnit 3.

Failure on 2025-01-10 As seen in Fig. 4.16, this failure is primarily driven by positive contributions from features such as a low `DS_Position`, specific low ranges for `Box106_EffX_Min`

Table 4.10: LIME Rules for Severe Failure on 2025-01-07 in PUnit 3

Feature	Rule (Condition)
CS_Torque_Max	$-0.61 <$ CS_Torque_Max ≤ -0.03
Box106_OutY_Avg	$-0.63 <$ Box106_OutY_Avg ≤ 0.01
Box108_EffX_Min	Box108_EffX_Min > 0.57
Box104_OutY_Max	Box104_OutY_Max > 0.58

and `Preheater_Temp`, a low `LI_Position`, and `Box104_EffX_Min` within a defined range. These conditions indicate a deviation in operational parameters that collectively signal an impending issue. In contrast, `Box106_EffX_Avg` and `ChamberPressure_2_Avg` show negative contributions, indicating their values were consistent with normal operation for this specific failure. All failure's contributing feature rules are summarized in Table 4.11.

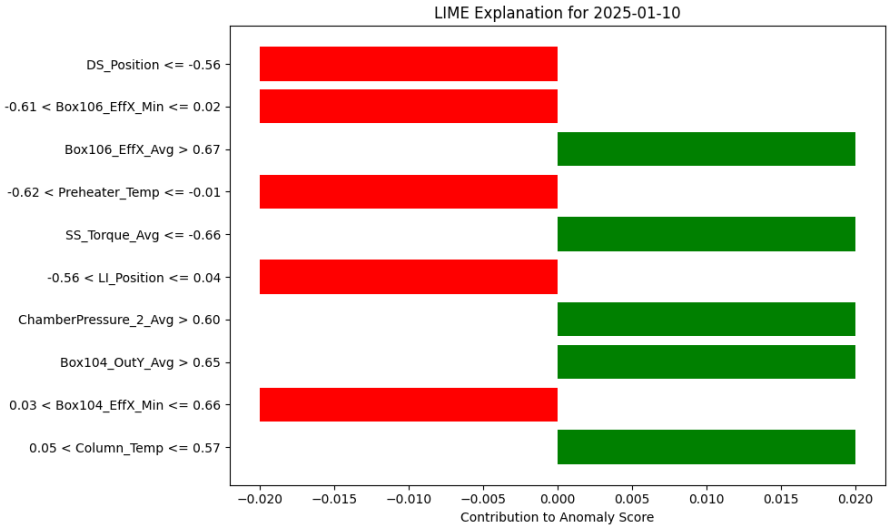


Figure 4.16: LIME explanation for severe failure on 2025-01-10 detected by `wae-gan-punit3-3` in PUnit 3.

Table 4.11: LIME Rules for Severe Failure on 2025-01-10 in PUnit 3

Feature	Rule (Condition)
DS_Position	DS_Position ≤ -0.56
Box106_EffX_Min	$-0.61 <$ Box106_EffX_Min ≤ -0.02
Preheater_Temp	$-0.62 <$ Preheater_Temp ≤ -0.01
Box104_EffX_Min	$0.03 <$ Box104_EffX_Min ≤ 0.66
LI_Position	$-0.56 <$ LI_Position ≤ 0.04

Failure on 2025-01-16 With Fig. 4.17 we can see that this failure is characterized by positive contributions from a low `CS_Torque_Max`, high `Box104_OutY_Avg`, low `Box105_OutY_Min`, a low `Preheater_Temp`, and `ChamberPressure_2_Max` within a specific range. These indicate distinct sensor behaviors contributing to the anomaly. Features such as `Box107_EffX_Avg`

and `ChamberPressure_1_Min` exhibit negative contributions, indicating their values were consistent with normal operation for this specific event. The detailed feature rules and their contributions to this failure are summarized in Table 4.12.

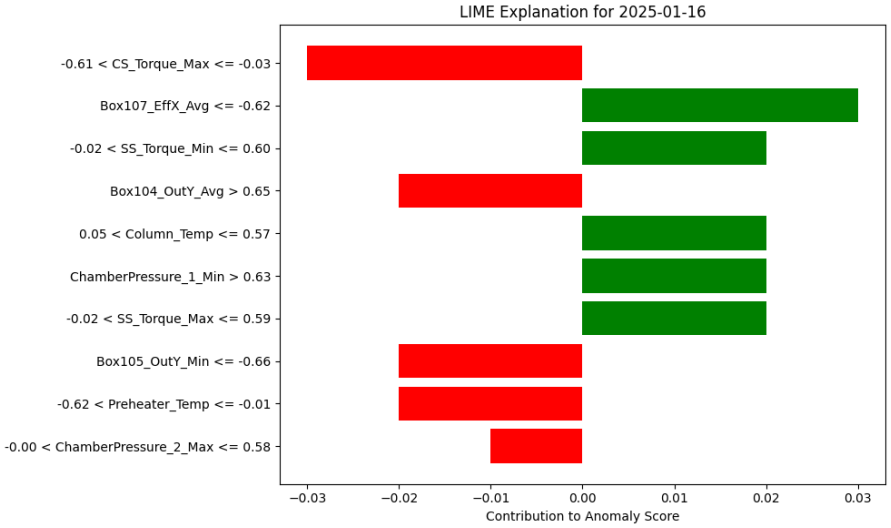


Figure 4.17: LIME explanation for severe failure on 2025-01-16 detected by `wae-gan-punit3-3` in PUnit 3.

Table 4.12: LIME Rules for Severe Failure on 2025-01-16 in PUnit 3

Feature	**Rule (Condition)**
CS_Torque_Max	$-0.61 < \texttt{CS\_Torque\_Max} \leq -0.56$
Box104_OutY_Avg	$\text{Box104\_OutY\_Avg} \leq -0.62$
Box105_OutY_Min	$\text{Box105\_OutY\_Min} \leq -0.66$
Preheater_Temp	$-0.62 < \text{Preheater\_Temp} \leq -0.01$
ChamberPressure_2_Max	$-0.62 < \text{ChamberPressure\_2\_Max} \leq -0.01$

4.3.2 PUnit 10

This section details the LIME explanations for severe failures detected by the optimal `wae-gan-punit10-4` model in PUnit 10. These insights are critical for identifying system-specific deviations.

Failure on 2025-01-07 Fig. 4.18 prominently highlights that this instance is primarily explained by positive contributions from `Box105_OutY_Min`, `CS_Torque_Max`, `Box108_OutY_Min`, and `DS_Torque_Min` being in specific low ranges, along with distinct conditions in `Box106_OutY_Min`. These indicate problematic mechanical or operational states. Conversely, features like `Chamber_Temp` and `Detector_Temp` show high negative contributions, indicating their values were aligned with normal operation for this specific failure. The detailed feature rules that contributed to this failure are summarized in Table 4.13.

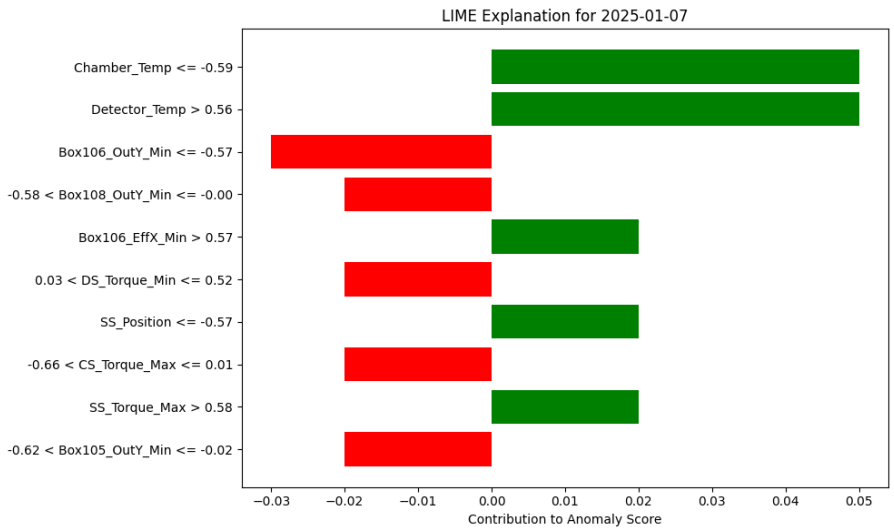


Figure 4.18: LIME explanation for severe failure on 2025-01-07 detected by `wae-gan-puni10-4` in PUnit 10.

Table 4.13: LIME Rules for Severe Failure on 2025-01-07 in PUnit 10

Feature	**Rule (Condition)**
Box106_OutY_Min	$\text{Box106\_OutY\_Min} \leq -0.57$
Box108_OutY_Min	$-0.58 < \text{Box108\_OutY\_Min} \leq -0.00$
DS_Torque_Min	$0.03 < \text{DS\_Torque\_Min} \leq 0.52$
CS_Torque_Max	$-0.66 < \text{CS\_Torque\_Max} \leq 0.01$
Box105_OutY_Min	$-0.62 < \text{Box105\_OutY\_Min} \leq -0.02$

Failure on 2025-01-10 As seen in Fig. 4.19, the failure is primarily driven by positive contributions from `DS_Torque_Avg`, `LI_Torque_Min`, `CS_Torque_Avg`, `Box104_EffX_Min`, and `Box107_OutY_Min`. These features point to specific torque and efficiency deviations. In contrast, features like `Column_Temp` and `Chamber_Temp` show strong negative contributions, indicating their values were consistent with normal operation for this specific failure. All failure's contributing feature rules are summarized in Table 4.14.

Table 4.14: LIME Rules for Severe Failure on 2025-01-10 in PUnit 10

Feature	**Rule (Condition)**
DS_Torque_Avg	$-0.56 < \text{DS\_Torque\_Avg} \leq 0.00$
LI_Torque_Min	$\text{LI\_Torque\_Min} \leq -0.60$
CS_Torque_Avg	$0.03 < \text{CS\_Torque\_Avg} \leq 0.57$
Box104_EffX_Min	$\text{Box104\_EffX\_Min} > 0.61$
Box107_OutY_Min	$-0.56 < \text{Box107\_OutY\_Min} \leq -0.02$

Failure on 2025-01-13 Fig. 4.20 illustrates that this failure is characterized by positive contributions from `Box107_EffX_Avg`, `Box106_OutY_Min`, `Box104_EffX_Avg`, and `Box108_OutY_Min`.

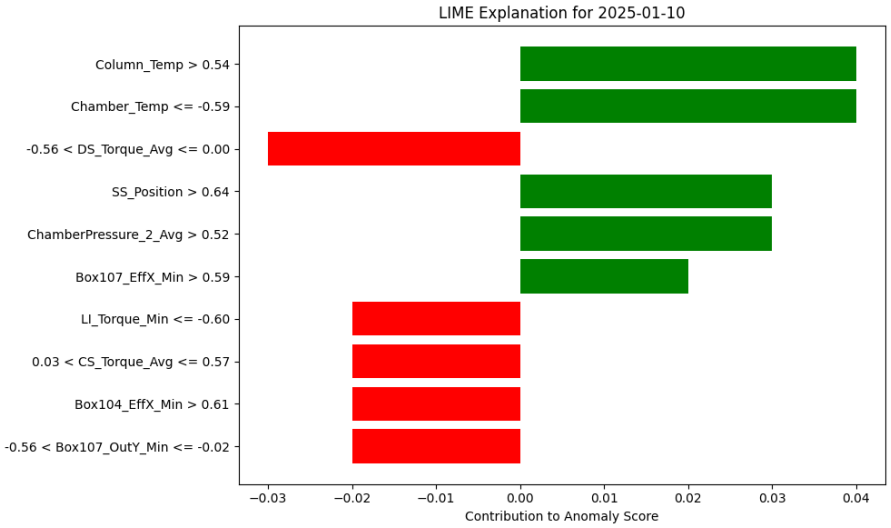


Figure 4.19: LIME explanation for severe failure on 2025-01-10 detected by `wae-gan-punit10-4` in PUnit 10.

These highlight specific efficiency and output anomalies. Conversely, features such as `ChamberPressure_2_Avg` and `Box105_OutY_Min` exhibit negative contributions, indicating their values were consistent with normal operation for this specific event. All failure's contributing feature rules are summarized in Table 4.15.

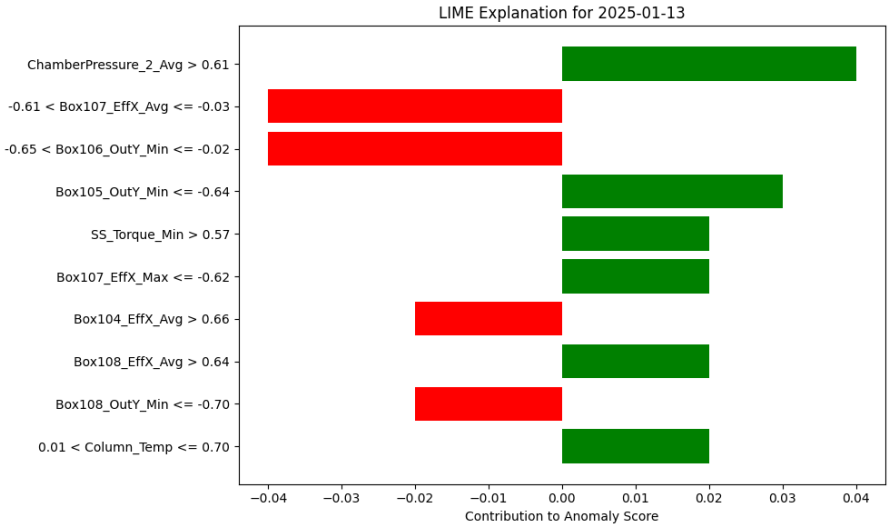


Figure 4.20: LIME explanation for severe failure on 2025-01-13 detected by `wae-gan-punit10-4` in PUnit 10.

Table 4.15: LIME Rules for Severe Failure on 2025-01-13 in PUnit 10

Feature	Rule (Condition)
Box107_EffX_Avg	$-0.61 <$ Box107_EffX_Avg ≤ -0.03
Box106_OutY_Min	$-0.65 <$ Box106_OutY_Min ≤ -0.02
Box104_EffX_Avg	Box104_EffX_Avg > 0.66
Box108_OutY_Min	Box108_OutY_Min ≤ -0.70

4.3.3 PUnit 12

This section details the LIME explanations for severe failures detected by the optimal `wae-gan-punit12-4` model in PUnit 12.

Failure on 2025-01-14 Fig. 4.21 prominently highlights that this instance is primarily explained by positive contributions from `LI_Position`, `Box108_OutY_Avg`, `Box105_OutY_Min`, and `Box108_EffX_Avg`. These features indicate specific positional, output, and efficiency anomalies. Oppositely, features like `Box104_EffX_Max` and `Box105_EffX_Avg` show negative contributions, indicating their values were aligned with normal operation for this specific failure. The detailed feature rules that contributed to this failure are summarized in Table 4.16.

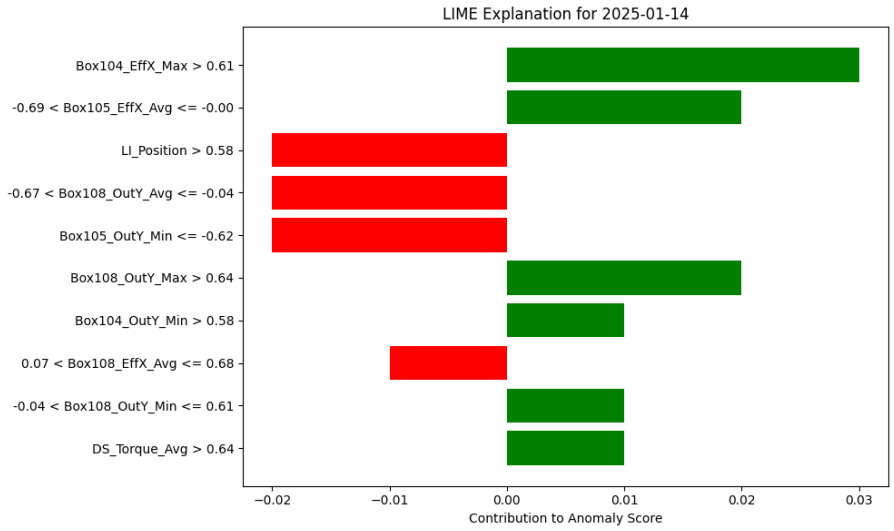


Figure 4.21: LIME explanation for severe failure on 2025-01-14 detected by `wae-gan-punit12-4` in PUnit 12.

Table 4.16: LIME Rules for Severe Failure on 2025-01-14 in PUnit 12

Feature	Rule (Condition)
LI_Position	LI_Position > 0.58
Box108_OutY_Avg	$-0.67 <$ Box108_OutY_Avg ≤ -0.04
Box105_OutY_Min	Box105_OutY_Min ≤ -0.62
Box108_EffX_Avg	$0.07 <$ Box108_EffX_Avg ≤ 0.68

4.3.4 PUnit 15

This section details the LIME explanations for severe failures detected by the optimal `wae-gan-punit15-4` model in PUnit 15. These insights are essential for pinpointing specific operational deviations in this unit.

Failure on 2025-01-08 Fig. 4.22 highlights that the anomaly was primarily influenced by a high maximum torque on the downstream shaft and elevated `Column_Temp`, both contributing negatively to normal behavior, suggesting mechanical strain or thermal issues. In contrast, features like `ChamberPressure_2_Min`, `Box105_EffX_Min`, and `Preheater_Temp` supported the anomaly score with positive contributions. The features that contributed to this failure and their conditions are summarized in Table 4.17.

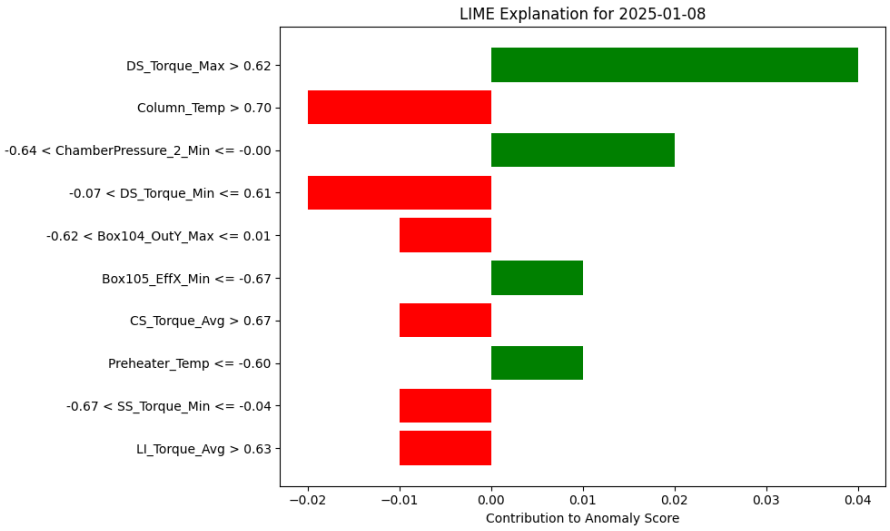


Figure 4.22: LIME explanation for severe failure on 2025-01-08 detected by `wae-gan-punit15-4` in PUnit 15.

Failure on 2025-01-12 As shown in Fig. 4.23, the most significant contributor to this anomaly was an elevated `ChamberPressure_1_Avg`. This suggests an abnormal pressure condition. Positive contributors also included `Box108_OutY_Avg` and multiple efficiency

Table 4.17: LIME explanation for severe failure on 2025-01-08 in PUnit 15

Feature	Rule (Condition)
Column_Temp	Column_Temp > 0.70
DS_Torque_Min	$-0.07 <$ DS_Torque_Min ≤ 0.61
Box104_OutY_Max	$-0.62 <$ Box104_OutY_Max ≤ 0.01
CS_Torque_Avg	CS_Torque_Avg > 0.67
SS_Torque_Min	$-0.67 <$ SS_Torque_Min ≤ -0.04
LI_Torque_Avg	LI_Torque_Avg > 0.63

indicators, pointing to other affected operational aspects. The features that contributed to this are summarized in Table 4.18.

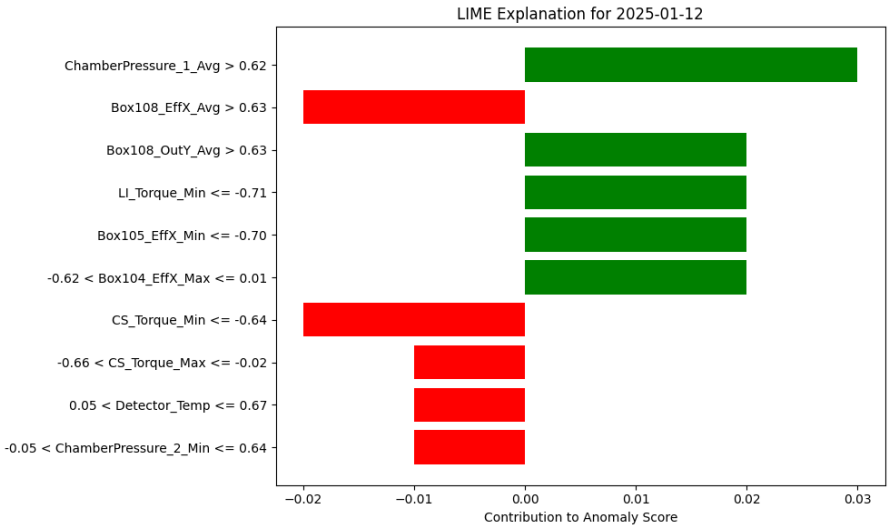


Figure 4.23: LIME explanation for severe failure on 2025-01-12 detected by `wae-gan-punit15-4` in PUnit 15.

Table 4.18: LIME explanation for severe failure on 2025-01-12 in PUnit 15

Feature	Rule (Condition)
Box108_EffX_Avg	Box108_EffX_Avg > 0.63
CS_Torque_Min	CS_Torque_Min ≤ -0.64
CS_Torque_Max	$-0.66 <$ CS_Torque_Max ≤ -0.02
Detector_Temp	$0.05 <$ Detector_Temp ≤ 0.67
ChamberPressure_2_Min	$-0.05 <$ ChamberPressure_2_Min ≤ 0.64

4.4. Overall Analysis

The experimental study detailed in this chapter successfully demonstrated the efficacy of the failure prediction system coupled with an explainability mechanism for proactive maintenance in complex industrial systems like the NDTech 2.0. The comprehensive

evaluation of both LSTM-AE and WAE-GAN models clearly established the superior performance of the WAE-GAN architecture in detecting critical component failures, characterized by its robust F1-scores and consistent lead times. The adversarial training and inherent latent space regularization of the WAE-GAN proved instrumental in distinguishing subtle anomalies from normal operational variations.

Crucially, the integration of the LIME framework transforms mere AD into actionable intelligence. By providing local, interpretable explanations for detected failures, LIME bridges the gap between complex model outputs and practical maintenance needs. The ability to identify specific sensor conditions and feature value ranges that contribute most significantly to an anomaly empowers maintenance personnel to rapidly diagnose root causes, prioritize interventions, and execute targeted repairs. This moves beyond a reactive maintenance paradigm, enabling truly predictive and preventive actions.

The consistent lead times for failure prediction (ranging from 30 to 70 minutes) across different PUnits further underscore the practical utility of this system. Such early warnings provide a critical window for intervention, significantly reducing the risk of catastrophic failures, minimizing downtime, and optimizing operational efficiency. This dual-layered approach—robust failure prediction complemented by clear, actionable explanations—represents a significant step towards a more intelligent and proactive maintenance strategy for the NDTech 2.0 system.

5. Conclusion

This thesis successfully delivered a sophisticated PdM system, augmented with robust XAI capabilities, specifically engineered for the intricate NDTech 2.0 cork quality control system. The foundational objectives - to move beyond traditional reactive maintenance by enabling early failure prediction and providing actionable insights into their root causes, have been comprehensively achieved. The outcome is a resilient PdM system delivering both precise failure anticipation and meaningful, interpretable explanations.

5.1. Contributions

A central innovation of this research lies in the development and validation of a novel two-layered architecture: a Failure Prediction Layer and an Explanation Layer. This modular and synergistic design proved exceptionally effective, culminating in a cohesive and powerful diagnostic system. Within the Failure Prediction Layer, an extensive exploration and implementation of DL-based autoencoders, specifically LSTM-AE and WAE-GAN models, were undertaken. Our findings definitively established the superior performance of the WAE-GAN architecture in detecting complex failure patterns, driven by its inherent architectural stability and efficient training methodology, making it particularly well-suited for the dynamic characteristics of NDTech 2.0 data.

A key practical contribution is the system's remarkable predictive capability: the optimized models consistently anticipate genuine failures within a critical lead time ranging from 30 to 70 minutes before their actual occurrence across various PUnits. This early warning capacity is paramount for enabling proactive interventions, thereby significantly minimizing downtime, mitigating potential operational disruptions, and enhancing safety in a critical industrial environment.

Furthermore, the integration of the Explanation Layer represents a significant advance. The insights derived from the LIME framework, presented as interpretable rules or conditions for each detected pre-failure event, provide unprecedented transparency. These detailed explanations empower maintenance teams with critical information, enabling them to precisely understand the specific sensor behaviors that contribute to an impending issue. This transparency is fundamental not only for guiding highly targeted maintenance tasks and optimizing resource allocation but also for fostering trust and adoption of automated predictive systems.

5.2. Limitations and Future Work

While this study marks substantial progress, it also naturally uncovers avenues for future research. Exploring alternative and emerging DL architectures, such as Transformers and GNNs, holds considerable promise for further enhancing prediction accuracy and efficiency, particularly given their strengths in handling complex sequential and relational data. Additionally, despite the promising results from the XAI component, the effectiveness of DL models remains intrinsically linked to the availability of extensive labeled data. Future work in the XAI domain should continue to explore methodologies that can provide even more comprehensive and robust explanations, especially as the complexity of the underlying detection models potentially increases. Addressing challenges related to data scarcity and developing techniques resilient to limited ground truth information will be paramount for advancing this critical field.

In summary, this thesis represents a transformative step towards optimizing control procedures and dramatically enhancing operational efficiency within the NDTech 2.0 system. By successfully combining cutting-edge PdM with interpretable XAI, we have developed a system poised to revolutionize current maintenance practices, enabling more proactive, data-driven, informed, and ultimately, more effective interventions. The profound insights gleaned from operational data, coupled with the system's powerful predictive and explanatory capabilities, underscore the immense value of leveraging advanced analytical tools for strategic decision-making in demanding industrial environments.

References

[1] J. Gama and A. Jorge, “The new spring for artificial intelligence,” *INESC TEC Science & Society*, vol. 1, no. 1, p. 26–29, December 2020. [Online]. Available: https://science-society.inesctec.pt/index.php/inesctecesociedade/article/view/27 [Cited on page 1.]

[2] J. Maclure, “The new ai spring: a deflationary view,” *AI & society*, vol. 35, no. 3, pp. 747–750, 2020. [Online]. Available: https://doi.org/10.1007/s00146-019-00912-z

[3] P. Parviainen, M. Tihinen, J. Kääriäinen, and S. Teppola, “Tackling the digitalization challenge: how to benefit from digitalization in practice,” *International journal of information systems and project management*, vol. 5, no. 1, pp. 63–77, 2017. [Online]. Available: https://doi.org/10.12821/ijispm050104

[4] C. Humby, “Data is the new oil,” *Proc. ANA Sr. Marketer’s Summit. Evanston, IL, USA*, vol. 1, 2006. [Cited on page 1.]

[5] J.-P. Hong, E.-J. Kim, and H.-Y. Park, “An analysis of determinants for artificial intelligence industry competitiveness,” *Journal of the Korea Institute of Information and Communication Engineering*, vol. 21, no. 4, pp. 663–671, 2017. [Online]. Available: https://doi.org/10.6109/jkiice.2017.21.4.663 [Cited on page 1.]

[6] B. Lee, B. Kim, and U. V. Ivan, “Enhancing the competitiveness of ai technology-based startups in the digital era,” *Administrative Sciences*, vol. 14, no. 1, 2024. [Online]. Available: https://www.mdpi.com/2076-3387/14/1/6 [Cited on page 1.]

[7] P. Jorzik, S. P. Klein, D. K. Kanbach, and S. Kraus, “Ai-driven business model innovation: A systematic review and research agenda,” *Journal of Business Research*, vol. 182, p. 114764, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0148296324002686 [Cited on page 1.]

[8] O. A. Farayola, A. A. Abdul, B. O. Irabor, and E. C. Okeleke, “Innovative business models driven by ai technologies: a review,” *Computer Science & IT Research Journal*, vol. 4, no. 2, pp. 85–110, 2023. [Online]. Available: https://www.academia.edu/download/109333870/774.pdf [Cited on page 1.]

[9] T. Zonta, C. A. da Costa, R. da Rosa Righi, M. J. de Lima, E. S. da Trindade, and G. P. Li, "Predictive maintenance in the industry 4.0: A systematic literature review," *Computers & Industrial Engineering*, vol. 150, p. 106889, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835220305787 [Cited on pages 1 and 16.]

[10] J. L. C. Bárcena, M. Daole, P. Ducange, F. Marcelloni, A. Renda, F. Ruffini, and A. Schiavo, "Fed-xai: Federated learning of explainable artificial intelligence models." in *XAI. it@ AI* IA*. Udine, 2022, pp. 104–117. [Online]. Available: https://ceur-ws.org/Vol-3277/paper8.pdf [Cited on pages viii, 1, and 6.]

[11] G7 Information and Communication Technologies Ministers' Meeting, "G7 ICT and Industry Ministers' Declaration," Tallinn, Estonia, September 2017, accessed March 27, 2025. [Online]. Available: https://ccdcoe.org/uploads/2018/11/G7-170926-ICT_Industry_Ministers_Declaration.pdf [Cited on page 1.]

[12] European Commission, "Ethics guidelines for trustworthy ai," European Commission, April 2019, accessed March 27, 2025. [Online]. Available: https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai [Cited on page 1.]

[13] F. K. Došilović, M. Brčić, and N. Hlupić, "Explainable artificial intelligence: A survey," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 0210–0215. [Online]. Available: https://doi.org/10.23919/MIPRO.2018.8400040 [Cited on page 1.]

[14] J. Jakubowski, N. Wojak-Strzelecka, R. P. Ribeiro, S. Pashami, S. Bobek, J. Gama, and G. J. Nalepa, "Artificial intelligence approaches for predictive maintenance in the steel industry: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2405.12785 [Cited on page 1.]

[15] R. Mobley, *An Introduction to Predictive Maintenance*, ser. Plant Engineering. Butterworth-Heinemann, 2002. [Online]. Available: https://books.google.pt/books?id=SjqXzxpAzSQC [Cited on page 16.]

[16] S. Selcuk, "Predictive maintenance, its implementation and latest trends," *Proceedings of the Institution of Mechanical Engineers, Part B*, vol. 231, no. 9, pp. 1670–1679, 2017. [Online]. Available: https://doi.org/10.1177/0954405415601640 [Cited on page 1.]

[17] A. Esteban Toscano, “Incremental decision tree models in data stream applied to predictive maintenance,” 2024, accessed March 27, 2025. [Online]. Available: https://helvia.uco.es/handle/10396/30018 [Cited on pages 1 and 12.]

[18] S. A. Amorim Cork, “History of cork,” accessed: March 29, 2025. [Online]. Available: https://www.amorim.com/en/cork/history/ [Cited on page 2.]

[19] V. Portugal, “The cork route,” accessed: March 29, 2025. [Online]. Available: https://www.visitportugal.com/en/content/the-cork [Cited on page 2.]

[20] M. L. Álvarez-Rodríguez, E. Recio, and J. J. R. Coque, “The analysis of natural cork stoppers in transversal sections as an effective tool to determine the origin of the taint by 2, 4, 6-trichloroanisole,” *European Food Research and Technology*, vol. 230, pp. 135–143, 2009. [Online]. Available: https://doi.org/10.1007/s00217-009-1153-6 [Cited on page 2.]

[21] A. Tarasov, M. Cabral, C. Loisel, P. Lopes, C. Schuessler, and R. Jung, “State-of-the-art knowledge about 2, 4, 6-trichloroanisole (tca) and strategies to avoid cork taint in wine,” in *Grapes and Wine*. IntechOpen, 2022. [Online]. Available: https://doi.org/10.5772/intechopen.103709 [Cited on page 2.]

[22] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. S. Awwal, and V. K. Asari, “A state-of-the-art survey on deep learning theory and architectures,” *Electronics*, vol. 8, no. 3, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/3/292 [Cited on pages viii, 4, and 5.]

[23] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608014002135

[24] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: https://www.nature.com/articles/nature14539 [Cited on pages 5 and 10.]

[25] H. Jiang, *Machine Learning Fundamentals: A Concise Introduction*. Cambridge University Press, 2021. [Online]. Available: https://books.google.pt/books?id=L2dNEAAAQBAJ [Cited on pages viii and 5.]

[26] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2, accessed May 02, 2025. [Online]. Available:

https://synapse.koreamed.org/pdf/10.4258/hir.2016.22.4.351 [Cited on pages 5, 6, and 10.]

[27] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4. [Online]. Available: https://www.worldcat.org/oclc/71008143 [Cited on page 6.]

[28] O. Janssens, V. Slavkovikj, B. Vervisch, K. Stockman, M. Loccufier, S. Verstockt, R. Van de Walle, and S. Van Hoecke, "Convolutional neural network based fault detection for rotating machinery," *Journal of Sound and Vibration*, vol. 377, pp. 331–345, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022460X16301638 [Cited on page 6.]

[29] L. Wen, X. Li, L. Gao, and Y. Zhang, "A new convolutional neural network-based data-driven fault diagnosis method," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 7, pp. 5990–5998, 2018. [Online]. Available: https://doi.org/10.1109/TIE.2017.2774777

[30] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data mining and knowledge discovery*, vol. 33, no. 4, pp. 917–963, 2019. [Online]. Available: https://doi.org/10.1007/s10618-019-00619-1 [Cited on page 6.]

[31] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735 [Cited on page 6.]

[32] Preeti, A. Dagar, R. Bala, and R. P. Singh, "Financial time series forecasting using deep learning network," in *Applications of Computing and Communication Technologies*, G. C. Deka, O. Kaiwartya, P. Vashisth, and P. Rathee, Eds. Singapore: Springer Singapore, 2018, pp. 23–33. [Online]. Available: https://doi.org/10.1007/978-981-13-2035-4_3

[33] P. Malhotra, L. Vig, G. Shroff, P. Agarwal *et al.*, "Long short term memory networks for anomaly detection in time series," in *Proceedings*, vol. 89, no. 9, 2015, p. 94. [Online]. Available: https://i6doc.com/en/book/?ISBN=9782875870155

[34] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014. [Online]. Available: https://arxiv.org/abs/1409.3215

[35] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 387–395. [Online]. Available: https://doi.org/10.1145/3219819.3219845 [Cited on page 6.]

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf [Cited on page 6.]

[37] B. Lim, S. O. Arık, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207021000637 [Cited on page 6.]

[38] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, accessed May 07, 2025. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2017/papers/Monti_Geometric_Deep_Learning_CVPR_2017_paper.pdf [Cited on page 6.]

[39] D. P. Kingma, M. Welling *et al.*, "Auto-encoding variational bayes," 2013, accessed May 07, 2025. [Online]. Available: http://web2.cs.columbia.edu/~blei/fogm/2018F/materials/KingmaWelling2013.pdf [Cited on page 6.]

[40] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/f033ed80deb0234979a61f95710dbe25-Paper.pdf

[41] J. An and S. Cho, “Variational autoencoder based anomaly detection using reconstruction probability,” *Special lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015, accessed May 08, 2025. [Online]. Available: http://dm.snu.ac.kr/static/docs/TR/SNUDM-TR-2015-03.pdf [Cited on page 6.]

[42] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” 2018. [Online]. Available: https://arxiv.org/abs/1803.01271 [Cited on page 6.]

[43] O. Serradilla, E. Zugasti, J. Rodriguez, and U. Zurutuza, “Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects,” *Applied Intelligence*, vol. 52, no. 10, pp. 10 934–10 964, 2022. [Online]. Available: https://doi.org/10.1007/s10489-021-03004-y [Cited on page 6.]

[44] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *J. Mach. Learn. Res.*, vol. 13, no. null, p. 281–305, Feb. 2012. [Online]. Available: https://doi.org/10.5555/2188385.2188395 [Cited on page 6.]

[45] M. Ammar, A. Haleem, M. Javaid, R. Walia, and S. Bahl, “Improving material quality management and manufacturing organizations system through industry 4.0 technologies,” *Materials Today: Proceedings*, vol. 45, pp. 5089–5096, 2021, second International Conference on Aspects of Materials Science and Engineering (ICAMSE 2021). [Online]. Available: https://doi.org/10.1016/j.matpr.2021.01.585 [Cited on page 7.]

[46] M. Achouch, M. Dimitrova, K. Ziane, S. Sattarpanah Karganroudi, R. Dhouib, H. Ibrahim, and M. Adda, “On predictive maintenance in industry 4.0: Overview, models, and challenges,” *Applied Sciences*, vol. 12, no. 16, 2022. [Online]. Available: https://doi.org/10.3390/app12168081 [Cited on pages viii, 8, and 16.]

[47] N. Davari, B. Veloso, G. d. A. Costa, P. M. Pereira, R. P. Ribeiro, and J. Gama, “A survey on data-driven predictive maintenance for the railway industry,” *Sensors*, vol. 21, no. 17, 2021. [Online]. Available: https://doi.org/10.3390/s21175739 [Cited on pages viii and 8.]

[48] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, “Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai,” *Information Fusion*, vol. 58, pp.

82–115, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253519308103 [Cited on pages 10, 11, and 12.]

[49] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang, "Xai—explainable artificial intelligence," *Science Robotics*, vol. 4, no. 37, p. eaay7120, 2019. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.aay7120 [Cited on page 10.]

[50] Z. C. Lipton, "The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery." *Queue*, vol. 16, no. 3, pp. 31–57, 2018. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3236386.3241340 [Cited on page 11.]

[51] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (xai)," *IEEE Access*, vol. 6, pp. 52 138–52 160, 2018. [Cited on page 10.]

[52] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," 2017. [Online]. Available: https://arxiv.org/abs/1702.08608 [Cited on page 11.]

[53] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM Comput. Surv.*, vol. 51, no. 5, Aug. 2018. [Online]. Available: https://doi.org/10.1145/3236009 [Cited on page 11.]

[54] P. Gohel, P. Singh, and M. Mohanty, "Explainable ai: current status and future directions," 2021. [Online]. Available: https://arxiv.org/abs/2107.07045 [Cited on pages viii and 11.]

[55] M. T. Ribeiro, S. Singh, and C. 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*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1135–1144. [Online]. Available: https://doi.org/10.1145/2939672.2939778 [Cited on pages 11 and 76.]

[56] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus,

S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf [Cited on page 11.]

[57] M. Christoph, "Interpretable machine learning: A guide for making black box models explainable," 2020. [Online]. Available: http://dlib.hust.edu.vn/handle/HUST/24999 [Cited on pages 11 and 78.]

[58] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 267, pp. 1–38, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370218305988 [Cited on page 12.]

[59] A. Najjar, H. I. Ashqar, and A. Hasasneh, "Predictive maintenance of urban metro vehicles: Classification of air production unit failures using machine learning," *Available at SSRN*, 2023, accessed May 13, 2025. [Online]. Available: https://www.researchgate.net/publication/371035904_Predictive_Maintenance_of_Urban_Metro_Vehicles_Classification_of_Air_Production_U [Cited on page 12.]

[60] N. Davari, B. Veloso, R. P. Ribeiro, P. M. Pereira, and J. Gama, "Predictive maintenance based on anomaly detection using deep learning for air production unit in the railway industry," in *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, 2021, pp. 1–10. [Online]. Available: https://doi.org/10.1109/DSAA53316.2021.9564181 [Cited on page 12.]

[61] Y. Wang, X. Du, Z. Lu, Q. Duan, and J. Wu, "Improved lstm-based time-series anomaly detection in rail transit operation environments," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 9027–9036, 2022. [Online]. Available: https://doi.org/10.1109/TII.2022.3164087 [Cited on page 12.]

[62] J. Gama, R. P. Ribeiro, S. Mastelini, N. Davarid, and B. Veloso, "A neuro-symbolic explainer for rare events: A case study on predictive maintenance," 2024. [Online]. Available: https://arxiv.org/abs/2404.14455 [Cited on pages 12 and 27.]

[63] J. Gama, R. P. Ribeiro, S. Mastelini, N. Davari, and B. Veloso, "From fault detection to anomaly explanation: A case study on predictive maintenance," *Journal of Web Semantics*, vol. 81, p. 100821, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570826824000076 [Cited on pages 12 and 31.]

[64] M. E. P. Silva, B. Veloso, and J. Gama, “Predictive maintenance, adversarial autoencoders and explainability,” in *Machine Learning and Knowledge Discovery in Databases: Applied Data Science and Demo Track*, G. De Francisci Morales, C. Perlich, N. Ruchansky, N. Kourtellis, E. Baralis, and F. Bonchi, Eds. Cham: Springer Nature Switzerland, 2023, pp. 260–275. [Online]. Available: https://doi.org/10.1007/978-3-031-43430-3_16 [Cited on pages x, 12, 27, 31, 35, 74, and 76.]

[65] G. Li and J. J. Jung, “Deep learning for anomaly detection in multivariate time series: Approaches, applications, and challenges,” *Information Fusion*, vol. 91, pp. 93–102, 2023. [Online]. Available: https://doi.org/10.1016/j.inffus.2022.10.008 [Cited on page 12.]

[66] Y. He and J. Zhao, “Temporal convolutional networks for anomaly detection in time series,” *Journal of Physics: Conference Series*, vol. 1213, no. 4, p. 042050, jun 2019. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1213/4/042050 [Cited on pages 12 and 75.]

[67] S. Han, X. Hu, H. Huang, M. Jiang, and Y. Zhao, “Adbench: Anomaly detection benchmark,” in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 32 142–32 159. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/cf93972b116ca5268827d575f2cc226b-Paper-Datasets_and_Benchmarks.pdf [Cited on page 12.]

[68] B. Hrnjica and S. Softic, “Explainable ai in manufacturing: A predictive maintenance case study,” in *Advances in Production Management Systems. Towards Smart and Digital Manufacturing*, B. Lalic, V. Majstorovic, U. Marjanovic, G. von Cieminski, and D. Romero, Eds. Cham: Springer International Publishing, 2020, pp. 66–73. [Online]. Available: https://doi.org/10.1007/978-3-030-57997-5_8 [Cited on page 13.]

[69] H. M. H. A. Alshkeili, S. J. Almheiri, and M. A. Khan, “Privacy-preserving interpretability: An explainable federated learning model for predictive maintenance in sustainable manufacturing and industry 4.0,” *AI*, vol. 6, no. 6, 2025. [Online]. Available: https://www.mdpi.com/2673-2688/6/6/117 [Cited on page 13.]

[70] O. Serradilla, E. Zugasti, J. Ramirez de Okariz, J. Rodriguez, and U. Zurutuza, "Adaptable and explainable predictive maintenance: Semi-supervised deep learning for anomaly detection and diagnosis in press machine data," *Applied Sciences*, vol. 11, no. 16, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/16/7376 [Cited on page 13.]

[71] G. Youness and A. Aalah, "An explainable artificial intelligence approach for remaining useful life prediction," *Aerospace*, vol. 10, no. 5, 2023. [Online]. Available: https://www.mdpi.com/2226-4310/10/5/474 [Cited on page 13.]

[72] K. Kobayashi and S. B. Alam, "Explainable, interpretable, and trustworthy ai for an intelligent digital twin: A case study on remaining useful life," *Engineering Applications of Artificial Intelligence*, vol. 129, p. 107620, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197623018043 [Cited on page 13.]

[73] M. L. Baptista, M. Mishra, E. Henriques, and H. Prendinger, "Using explainable artificial intelligence to interpret remaining useful life estimation with gated recurrent unit," *Annual Conference of the PHM Society*, vol. 16, no. 1, 2024. [Cited on page 13.]

[74] M. Jakobs, B. Veloso, and J. Gama, "Interpretable rules for online failure prediction: A case study on the metro do porto dataset," 2025. [Online]. Available: https://arxiv.org/abs/2502.07394 [Cited on page 13.]

[75] M. Kisten, A. E.-S. Ezugwu, and M. O. Olusanya, "Explainable artificial intelligence model for predictive maintenance in smart agricultural facilities," *IEEE Access*, vol. 12, pp. 24 348–24 367, 2024. [Online]. Available: https://doi.org/10.1109/ACCESS.2024.3365586 [Cited on page 13.]

[76] J. Chatterjee and N. Dethlefs, "Xai4wind: A multimodal knowledge graph database for explainable decision support in operations & maintenance of wind turbines," 2021. [Online]. Available: https://arxiv.org/abs/2012.10489 [Cited on page 13.]

[77] H. Je-Gal, Y.-S. Park, S.-H. Park, J.-U. Kim, J.-H. Yang, S. Kim, and H.-S. Lee, "Time-series explanatory fault prediction framework for marine main engine using explainable artificial intelligence," *Journal of Marine Science and Engineering*, vol. 12, no. 8, 2024. [Online]. Available: https://www.mdpi.com/2077-1312/12/8/1296 [Cited on page 13.]

[78] J. E. Jeon, S. J. Hong, and S.-S. Han, “Utilization of machine learning and explainable artificial intelligence (xai) for fault prediction and diagnosis in wafer transfer robot,” *Electronics*, vol. 13, no. 22, 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/22/4471 [Cited on page 13.]

[79] H. Wu, A. Huang, and J. W. Sutherland, “Layer-wise relevance propagation for interpreting lstm-rnn decisions in predictive maintenance,” *International Journal of Advanced Manufacturing Technology*, vol. 119, no. 1–2, p. 345–361, 2022. [Online]. Available: https://doi.org/10.1007/s00170-021-08765-2 [Cited on page 13.]

[80] J. W. Tukey *et al.*, *Exploratory data analysis*. Springer, 1977, vol. 2. [Cited on page 33.]

[81] H. Nguyen, K. Tran, S. Thomassey, and M. Hamad, “Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management,” *International Journal of Information Management*, vol. 57, p. 102282, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S026840122031481X [Cited on pages ix and 74.]

[82] P. Mobtahej, X. Zhang, M. Hamidi, and J. Zhang, “An lstm-autoencoder architecture for anomaly detection applied on compressors audio data,” *Computational and Mathematical Methods*, vol. 2022, no. 1, p. 3622426, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/3622426 [Cited on pages ix and 75.]

[83] A. Boukerche, L. Zheng, and O. Alfandi, “Outlier detection: Methods, models, and classification,” *ACM Comput. Surv.*, vol. 53, no. 3, Jun. 2020. [Online]. Available: https://doi.org/10.1145/3381028 [Cited on page 77.]

A. Model Concepts and Configuration Details

A.1. Conceptual Architectures for the Autoencoders Models

AEs are neural networks architected to learn compact, low-dimensional latent representations of input data. Their core mechanism involves an encoder that maps input X to a latent vector z, and a decoder that reconstructs the original input $\hat{X}$ from z. Anomalies are identified by quantifying the discrepancy between the original input and its reconstruction, typically measured by a high reconstruction error. We specifically focus on two advanced AE variants:

LSTM-AE It leverages the power of LSTM networks, which are exceptionally adept at capturing long-term temporal dependencies and patterns within sequential data. This makes them highly suitable for time-series AD in industrial contexts where system states evolve over time.

LSTM-AEs leverage the temporal modeling strength of LSTMs to encode and reconstruct time-series data, capturing long-term dependencies critical for AD. Fig. A.1 presents an illustration of the structure and the operational principle of a typical LSTM module. Unlike traditional RNNs, LSTMs incorporate a memory cell along with three key gates (input, forget, and output gates) that regulate the flow of information. These gates use activation functions (typically sigmoid and tanh) to determine what information should be retained, updated, or discarded at each time step. This gating mechanism allows LSTMs to effectively capture and retain relevant patterns, even across long sequences, making them particularly useful in tasks that depend on temporal dynamics such as prediction, classification, and AD.

This capacity to learn temporal dependencies while managing long-term information flow makes LSTMs an excellent backbone for autoencoder architectures aimed at modeling normal system behavior in time-series data.

As depicted conceptually in Fig. A.2 (inspired by typical LSTM-AE architectures), the LSTM-AE comprises:

- **Encoder:** An LSTM-based encoder processes the input time-series sequence $X = (x_1, x_2, \ldots, x_T)$. Each x_t represents the feature vector at time t. The encoder maps

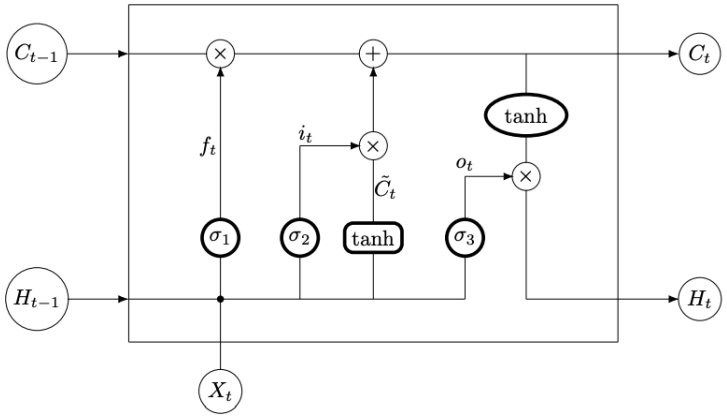


Figure A.1: A module of LSTM network [81].

this sequence into a compressed, fixed-dimensional latent representation z. This latent vector aims to encapsulate the essential characteristics of the normal temporal pattern.

- **Decoder:** An LSTM-based decoder takes the latent vector z as input and attempts to reconstruct the original time-series sequence $\hat{X} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_T)$.

The model is exclusively trained on data corresponding to the normal operation of each PUnit. The anomaly score for a given input sequence is calculated as the MSE between the input X and its reconstruction $\hat{X}$:

$$\text{MSE} = \frac{1}{T \cdot D} \sum_{t=1}^{T} \sum_{d=1}^{D} (x_{t,d} - \hat{x}_{t,d})^2$$

where T is the sequence length and D is the number of features. A higher MSE signifies a greater deviation from the learned normal patterns, hence indicating a potential anomaly.

WAE-GAN It integrates concepts from *Wasserstein Autoencoders* (WAE) and GANs to achieve a more robust and flexible AD capability. The WAE aims to match the aggregated posterior of the encoder with a prior distribution (e.g., Gaussian) in the latent space using the Wasserstein distance, leading to a more structured and disentangled latent representation compared to traditional AEs. The GAN component, specifically a discriminator, further enhances the quality of the generated samples by pushing the decoder to produce reconstructions that are indistinguishable from real data.

The architecture of a WAE-GAN for time-series data typically involves (as seen in Fig. A.3, inspired by [64]):

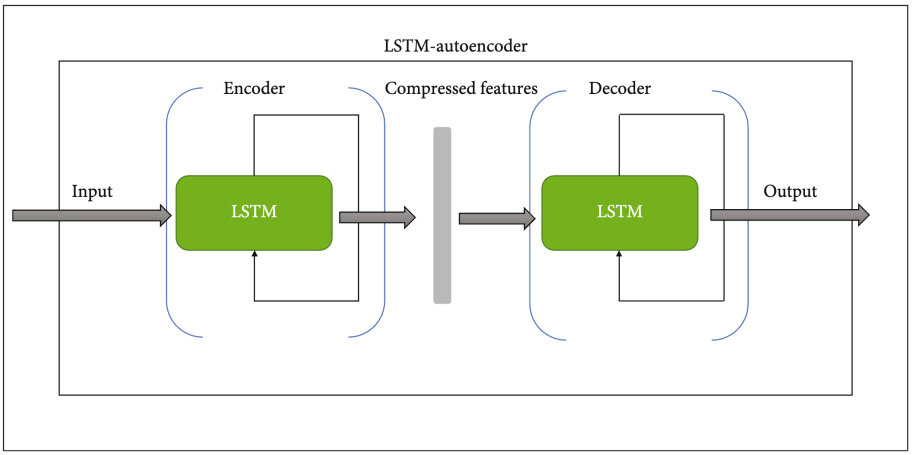


Figure A.2: Conceptual architecture of a LSTM-AE for time-series AD. The encoder maps an input sequence to a latent representation, which the decoder uses to reconstruct the sequence [82].

- **Encoder (E):** Maps input data X to a latent representation z.
- **Decoder (G):** Acts as a generator, taking a latent code z (sampled from the prior or from the encoder's output) and attempting to reconstruct the input $\hat{X}$.
- **Discriminator (D):** Distinguishes between real data X and reconstructed/generated data $\hat{X}$.

The training objective involves minimizing the reconstruction error (like in standard AEs) while simultaneously optimizing a GAN objective to ensure that the latent space is well-behaved and that the decoder can generate high-quality reconstructions. The anomaly score can be derived from the reconstruction error, potentially augmented by the discriminator's output, as anomalous inputs are expected to yield poor reconstructions and/or be easily identified as "fake" by the discriminator.

Integration of TCN in WAE-GAN For handling sequential data effectively within the WAE-GAN framework, we will incorporate TCNs as the backbone for both the encoder and decoder components. TCNs, as described in works like [66], utilize dilated causal convolutions, allowing them to capture long-range dependencies efficiently with a fixed receptive field while maintaining causality (i.e., output at time t only depends on inputs at time t and before). This makes TCNs highly suitable for time-series data as they offer:

- **Causality:** Ensures that predictions at time t do not use any future information.

- **Variable Receptive Field:** Through dilated convolutions, they can effectively cover long sequences without using recurrent connections, reducing vanishing/exploding gradient problems.
- **Parameter Sharing:** Convolutional filters are shared across time steps, leading to more efficient learning.

By using TCNs within the encoder and decoder of the WAE-GAN, we aim to enhance the model's ability to learn temporal patterns and reconstruct time-series data more accurately for AD.

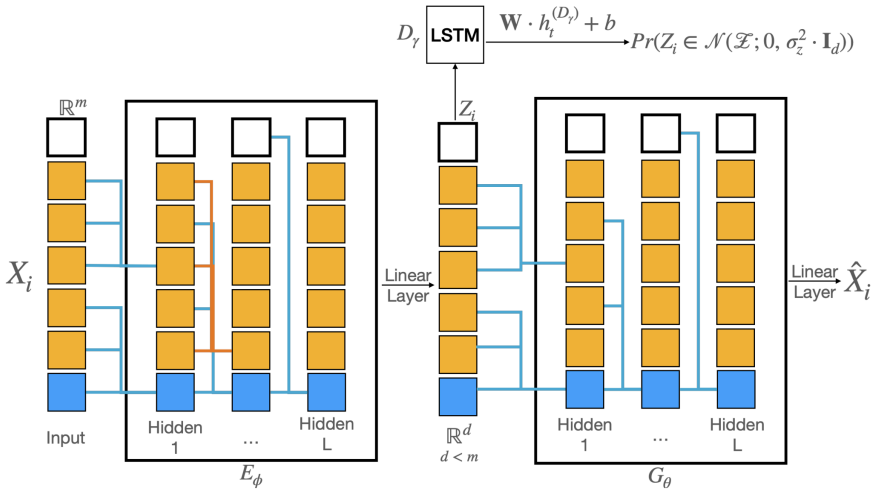


Figure A.3: Conceptual architecture of a WAE-GAN for AD, with TCN as encoder and decoder backbones. The encoder maps input to latent space, the decoder reconstructs, and the discriminator distinguishes real from reconstructed data [64].

For each of the four PUnits, dedicated instances of the LSTM-AE and WAE-GAN models will be independently trained on their respective normal operational data. This modular approach ensures that each PUnit's unique operational signature is accurately learned, leading to highly granular and precise AD. The output from this Failure Prediction Layer for each PUnit is a continuous anomaly score stream. When this score surpasses a pre-calibrated threshold, an alarm is activated for that specific PUnit, and the corresponding input features X and the anomaly score are passed to the Explanation Layer for further diagnostic analysis.

A.2. LIME: Local Interpretable Model-Agnostic Explanations

LIME [55], constitutes a methodology for interpreting predictions made by black-box ML models. Its main objective is to develop localized surrogate models, which are interpretable models trained to approximate the predictions of the underlying black-box model

for individual instances. Unlike global surrogate models, LIME focuses on explaining single predictions rather than providing a holistic model interpretation.

How LIME Works The fundamental concept of LIME revolves around understanding the logic behind a specific prediction by analyzing variations in the input data. This is achieved by generating a new dataset of perturbed samples and obtaining their corresponding predictions from the black-box model. Subsequently, local surrogate models, such as linear regression or *Decision Tree* (DT)s, are trained on this new dataset. The interpretability of the model is optimized locally, ensuring it accurately approximates the black-box model's predictions for a specific instance. Mathematically, LIME formulates this as an optimization task, balancing a loss function (measuring proximity to the original prediction) with model complexity.

To train the local surrogate model, LIME perturbs the dataset around the instance of interest and generates black-box model predictions for these perturbations. These new predictions are then assigned weights based on their proximity to the instance of interest. A weighted, interpretable model is then trained on this perturbed dataset, providing a localized approximation of the black-box model's predictions. LIME offers flexibility in choosing interpretable surrogate models (e.g., linear regression, DTs) and allows users to manage model complexity, such as by defining the maximum number of features. Simplicity is often prioritized for interpretability. The process involves selecting the number of features (K) and employing techniques like Lasso or forward/backward feature selection. Data perturbation generation depends on the data type: for tabular data, new samples are created by perturbing each feature individually, with perturbations drawn from a normal distribution based on the feature's mean and standard deviation. LIME thus provides a systematic approach to interpreting black-box predictions through localized approximations using interpretable surrogate models. As described in [83], LIME can improve interpretability and foster a deeper understanding of decision-making processes in complex ML models by focusing on individual instances.

Advantages of LIME LIME boasts several advantages. It is highly adaptable, supporting tabular, text, and image data, making it widely applicable across domains, for example, by identifying key contributing words in text or highlighting important image regions. Being model-agnostic, LIME can be applied to any ML model (e.g., DTs, ANNs, SVM), providing crucial versatility for practitioners who need to explain diverse models. Users also benefit

from the flexibility to choose interpretable surrogate models. Furthermore, LIME produces human-friendly explanations that are often succinct and contrastive, making it suitable for audiences with limited time or technical expertise. The fidelity measure provides valuable insight into the reliability of the interpretable model's approximation within the instance's neighborhood. Additionally, explanations can incorporate other interpretable features not used in the original model's training, provided they are derived from the data instances themselves.

Limitations of LIME On the other hand, LIME presents certain disadvantages. A significant challenge with tabular data is the precise characterization of the neighborhood, which in [57] it's identified as its most substantial obstacle. This necessitates careful experimentation with diverse kernel configurations and a subjective assessment of explanation intelligibility for each application. While local explanations are valuable, they may fail to capture the overall model behavior, particularly with complex or highly non-linear decision boundaries. The explanation model's complexity must be predetermined, requiring the user to balance fidelity and sparsity optimally. A more critical issue is the instability of explanations, as closely situated points can yield considerably disparate explanations. LIME struggles to define a meaningful neighborhood for rare events, making it less reliable for outlier detection or highly imbalanced data distributions. Moreover, LIME is susceptible to manipulation to overcome biases, potentially diminishing trust in its explanations. It is also not adequate for complete attributions, making it less suitable for compliance scenarios where full prediction explanations might be legally required or for debugging where all reasons are preferred.

A.3. Models Hyperparameter Tuning

The architectural parameters, including hidden dimension, dropout, LSTM layers, and latent vector dimension, were subjected to rigorous tuning for each specific PUnit (3, 10, 12, 15) to ensure optimal performance and effective generalization, while number of features remained constant at 56 and number of embeddings at 8 across all models. Table A.1 provides a detailed breakdown of the LSTM-AE model hyperparameters for each PUnit.

The selection of hidden dimension was primarily driven by empirical testing and established heuristics concerning dataset scale, aiming to balance model capacity with the volume of available data. The latent space dimension (either 14 or 19) was strategically chosen to effectively encapsulate the salient characteristics of distinct feature groups within

Table A.1: LSTM-AE Hyperparameters for each PUnit

Punit 3				
Model ID	**Hidden Dim.**	**Dropout**	**LSTM Layers**	**Latent Dim.**
lstm-ae-punit3-1	40	0.10	3	14
lstm-ae-punit3-2	56	0.15	3	14
lstm-ae-punit3-3	56	0.10	2	14
Punit 10				
Model ID	**Hidden Dim.**	**Dropout**	**LSTM Layers**	**Latent Dim.**
lstm-ae-punit10-1	40	0.15	3	14
lstm-ae-punit10-2	56	0.20	3	14
lstm-ae-punit10-3	32	0.15	2	14
Punit 12				
Model ID	**Hidden Dim.**	**Dropout**	**LSTM Layers**	**Latent Dim.**
lstm-ae-punit12-1	40	0.15	3	14
lstm-ae-punit12-2	56	0.20	2	14
lstm-ae-punit12-3	64	0.20	3	19
Punit 15				
Model ID	**Hidden Dim.**	**Dropout**	**LSTM Layers**	**Latent Dim.**
lstm-ae-punit15-1	56	0.10	2	14
lstm-ae-punit15-2	56	0.20	3	14
lstm-ae-punit15-3	64	0.20	3	19

the 56 continuous variables, such as `LI_Torque`, `Box104`, and `ChamberPressure` readings, with the optimal value determined through iterative empirical evaluation for each PUnit. Furthermore, the LSTM layers and dropout parameters were iteratively adjusted; notably, higher dropout rates were often paired with an increased number of LSTM layers to proficiently mitigate the risk of overfitting, especially given the considerable size of the dataset.

Table A.2 summarizes the Encoder/Decoder-specific hyperparameters used for each PUnit's WAE-GAN model. The discriminator hyperparameters are identical across models—matching the latent space dimension—with a fixed dropout rate of 0.4, while the Encoder/Decoder dropout rate are fixed at 0.2. As these values are consistent, they are omitted from the table.

Table A.2: WAE-GAN Hyperparameters for each PUnit

Punit 3				
Model ID	**Hidden Dim.**	**TCNs Layers**	**Latent Dim.**	**Kernel size**
wae-gan-punit3-1	32	4	19	4
wae-gan-punit3-2	56	3	14	4
wae-gan-punit3-3	40	5	14	3
wae-gan-punit3-4	32	5	19	3
Punit 10				
Model ID	**Hidden Dim.**	**TCNs Layers**	**Latent Dim.**	**Kernel size**
wae-gan-punit10-1	56	3	19	4
wae-gan-punit10-2	56	4	14	4
wae-gan-punit10-3	40	5	14	3
wae-gan-punit10-4	56	5	19	3
Punit 12				
Model ID	**Hidden Dim.**	**TCNs Layers**	**Latent Dim.**	**Kernel size**
wae-gan-punit12-1	32	4	19	4
wae-gan-punit12-2	56	4	14	4
wae-gan-punit12-3	40	5	14	3
wae-gan-punit12-4	32	5	19	3
Punit 15				
Model ID	**Hidden Dim.**	**TCNs Layers**	**Latent Dim.**	**Kernel size**
wae-gan-punit15-1	32	4	19	4
wae-gan-punit15-2	56	4	14	4
wae-gan-punit15-3	32	5	14	3
wae-gan-punit15-4	32	5	19	3

B. Detailed Model Architectures and Training

B.1. LSTM-AE

This section provides the comprehensive architectural details and training process for the LSTM-AE models implemented in this study.

Model Architecture Details

The model's input comprises two distinct data streams, continuous features and chambers, processed collaboratively to form a comprehensive input representation. As illustrated in Table 3.1, the continuous features, encompassing a diverse array of operational parameters from the NDTech 2.0 modules, such as torque, position, and various temperature metrics; chambers, on the other hand, provides categorical identifiers for 8 unique chambers within the system. To seamlessly integrate these discrete categorical variables into the continuous neural network computations, an embedding layer is employed. This layer transforms each chamber into a dense vector, with the chamber embedding dimension set to 3, a dimension derived from squared root of the number of chambers, and the number of embeddings is 8 which is the number of chambers. The continuous features are then concatenated with their corresponding learned chamber embeddings along the feature dimension, producing a unified input vector for the subsequent recurrent layers.

Encoder The encoder component is constructed as a multi-layered LSTM network. Its primary function is to process the concatenated input sequences and distill them into a compact, fixed-dimensional latent representation. The `nn.LSTM`* layers within the encoder are configured with `batch_first` set to True, ensuring input and output tensors are shaped as (`batch`, `sequence_length`, `features`). The input size of the LSTM is defined by the sum of the number of continuous features, which is 56, and the (as said before) chamber embedding dimension (3), totaling 59. The hidden size of the encoder's LSTM, which directly dictates the dimensionality of the learned latent space. The network's depth is controlled by LSTM layers, and a dropout rate is applied to mitigate overfitting during training. The final latent representation (latent vector) is obtained by taking the mean of the hidden states across all LSTM layers, thereby aggregating temporal information into a single, representative vector for each input sequence.

*https://docs.pytorch.org/docs/stable/generated/torch.nn.LSTM.html

Decoder The decoder component is designed to reconstruct the original input sequence from the latent vector generated by the encoder. Distinctively, the decoder's LSTM layers are configured as bidirectional, enabling the model to capture dependencies from both past and future contexts within the sequence, which typically leads to superior reconstruction quality. The input size of the decoder's LSTM is the latent vector dimension. This latent vector is first replicated to match the original sequence length before being fed as input to the decoder's LSTM. The hidden size of the decoder's LSTM is specified by hidden dimension. Subsequent to the bidirectional LSTM layers, a linear output layer (nn.Linear*) maps the concatenated forward and backward hidden states (resulting in twice hidden dimension features) to the reconstructed output. This output layer is meticulously designed to reconstruct both the continuous features and the chamber embeddings. The reconstructed continuous part aligns directly with the original number of features dimensionality. The reconstructed chambers, initially in chamber embedding dimension, are then passed through a dedicated linear layer to predict the original number embeddings (chamber categories), followed by a permutation operation to correctly align dimensions for the cross-entropy loss computation.

Training Process

The training of the LSTM-AE models was executed through a meticulous and robust process designed to imbue the models with strong failure detection capabilities. Each PUnit's dataset underwent a stratified division into training and validation sets, with an 85% / 15% split respectively, and shuffling was applied to ensure the stochasticity and representativeness of each subset. Data iteration during training and evaluation was efficiently managed by DataLoader† instances, employing a batch size of 32.

The fundamental training objective for the LSTM-AE is to minimize the discrepancy between the original input sequences and their reconstructed counterparts. This objective is achieved through the minimization of a sophisticated hybrid loss function, designed to account for both continuous and categorical features:

- **MSE:** This loss component is applied to the continuous features, quantifying the squared difference between the reconstructed input and the continuous features. It is calculated per sample, and then averaged across the sequence length and

*https://docs.pytorch.org/docs/stable/generated/torch.nn.Linear.html
†https://docs.pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader

feature dimensions, thereby imposing a penalty on inaccuracies in the numerical sensor reading reconstructions.

- **Cross-Entropy Loss:** This component addresses the categorical chamber features, comparing the transformed reconstructed chambers (predicted categorical probabilities) against the true chambers labels. This loss effectively quantifies the divergence between the model's categorical predictions and the ground truth.

The total loss for each training sample is computed as the sum of these MSE and Cross-Entropy components, which is then averaged across the entire batch to derive the final loss value for optimization.

Model optimization was conducted using the Adam optimizer*, initialized with a learning rate of $1e^{-3}$. To enhance training stability and foster superior performance, an adaptive learning rate scheduler, ReduceLROnPlateau†, was integrated into the training regimen. This scheduler dynamically monitors the validation loss and adjusts the learning rate by a factor, if no significant improvement in loss is observed over a patience window of epochs. A minimum learning rate of $1e^{-4}$ prevents the learning rate from decaying excessively, and a threshold of 0.03 defines the minimum improvement required to consider a validation loss reduction significant. Furthermore, to mitigate the common challenge of exploding gradients in recurrent neural networks, gradient clipping (nn.utils.clip_grad_norm_‡) was consistently applied during the backpropagation phase.

Each LSTM-AE model was trained for a total of 100—150 epochs. Within each epoch, the model iteratively processed batches from the training data loader, computed the combined loss, performed backpropagation to calculate gradients, and updated its internal weights via the optimizer. Both training and validation losses were meticulously logged after each epoch, enabling real-time observation of the model's convergence trajectory and prompt identification of potential overfitting. The learning rate was dynamically adjusted by the scheduler based on the training loss performance.

Upon the completion of the training phase, the AD threshold for each PUnit's LSTM-AE model was rigorously determined based on the statistical distribution of reconstruction errors computed for all normal data samples within the training dataset. As elaborated in Section 3.4.4.2, a well-known thresholding method was adopted to distinguish between

*https://docs.pytorch.org/docs/stable/generated/torch.optim.Adam.html
†https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html
‡https://docs.pytorch.org/docs/stable/generated/torch.nn.utils.clip_grad_norm_.html

normal and anomalous observations. After training, it was calculated the reconstruction losses across the training dataset, enabling the precise computation of Q1 and Q3. Any new incoming data point from the NDTech 2.0 system yielding a RE exceeding this empirically determined threshold is subsequently classified as an anomaly, providing a dynamic and statistically grounded mechanism for identifying deviations from normal operational behavior.

B.2. WAE-GAN

This section provides the comprehensive architectural details and training process for the WAE-GAN models, including the specifics of the TCN blocks.

Model Architecture Details

This model's input has the same initial assumptions as described in B.1

Temporal Block The Temporal Block serves as the fundamental building unit for both the TCN-based Encoder and Decoder. Each block meticulously processes sequential data through two convolutional layers (nn.Conv1d*, where weight normalization is applied to the convolutional weights for enhanced training stability (weight_norm†). These are sequentially followed by batch normalization (nn.BatchNorm1d‡), a *Rectified Linear Unit* (ReLU) activation function (nn.ReLU§), and a dropout layer (nn.Dropout¶). A critical aspect is the dilation parameter, which exponentially increases with layer depth, enabling the receptive field of the network to expand rapidly without a proportional increase in parameters, thereby efficiently capturing long-range dependencies across the time series. Padding is precisely calculated to maintain the sequence length throughout the convolutional operations. A residual connection is integrated, wherein the block's input is added to its output before the final activation, a mechanism vital for training deeper networks by mitigating vanishing gradients. A downsample 1x1 convolution is conditionally applied to the residual path if input and output channel dimensions differ, ensuring compatibility for the element-wise summation.

*https://docs.pytorch.org/docs/stable/generated/torch.nn.Conv1d.html
†https://docs.pytorch.org/docs/stable/generated/torch.nn.utils.weight_norm.html
‡https://docs.pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html
§https://docs.pytorch.org/docs/stable/generated/torch.nn.ReLU.html
¶https://docs.pytorch.org/docs/stable/generated/torch.nn.Dropout.html

Encoder The Encoder functions as the generative component within the AE framework, responsible for mapping the high-dimensional input data into a lower-dimensional latent space representation. The input streams comprise the continuous features (56 dimensions) and chambers (8 categorical identifiers). An Embedding layer transforms the discrete chambers into continuous chamber embedding dimension (3-dimensional) vectors. These embeddings are then concatenated with the continuous features, forming a unified input tensor of 59 dimensions. This combined input sequence is then transposed (permute) to align with the (batch size, channels, sequence length) format required by the 1D convolution layers within the TCN blocks. A series of Temporal Blocks (TCN layers) processes this permuted input. The design parameters, including number of layers and kernel size, were empirically chosen such that the TCN's receptive field is sufficiently large to encompass the known 8-timestep cycle inherent in the chamber data. A final Linear layer projects the aggregated TCN output to the latent space dimension.

Decoder The Decoder is tasked with reconstructing the original input sequence from the latent space vector produced by the Encoder. It receives the latent space and applies similar Temporal Block structures to expand this compressed representation back to the original sequence length and feature dimensions. The latent space vector is first permuted to the TCN's expected input format. The output of the TCN layers is then passed through an Linear layer to yield a combined reconstruction of stride (56 continuous features) and chamber embedding dimension (3 for chamber embeddings). The continuous features (reconstructed continuous) are directly extracted. For the chamber features, the embedded reconstruction is fed into another dedicated Linear layer to predict the number of embeddings (8) categorical probabilities, which are then permuted to facilitate cross-entropy loss computation.

Discriminator The LSTM-Discriminator operates within the latent space, designed to differentiate between latent vectors derived from the Encoder's output and those sampled from a standard multivariate Gaussian distribution. The discriminator utilizes a multi-layered LSTM network, processing the latent space vectors. The input dimension of the discriminator matches the latent space dimension of the Encoder. The hidden layer size of the LSTM is to capture intricate patterns within the latent vectors, considering the potential temporal dependencies or cyclical patterns within the latent embeddings. The final

hidden state of the LSTM is passed through an Linear layer followed by a sigmoid activation, producing a scalar probability indicating whether the input latent vector is perceived as ”real” or ”fake”.

Training Process

The training of the WAE-GAN models is a sophisticated multi-objective optimization problem. It involves iteratively training the Discriminator and the combined Encoder-Decoder (acting as the autoencoder/generator) to achieve a harmonious balance: the autoencoder aims to minimize RE and align its latent space with a prior distribution, while the Discriminator strives to differentiate between real and generated latent samples.

The training schedule is carefully orchestrated for each epoch, with the Encoder and Decoder being updated twice for every single update of the Discriminator. This specific ratio was determined through empirical testing to be optimal. This asymmetric training frequency helps in maintaining a balanced adversarial game, preventing the Discriminator from becoming overly powerful too quickly and allowing the Encoder and Decoder to sufficiently ”catch up” in their learning capabilities, thus promoting more stable convergence and avoiding issues like mode collapse.

The training process involves distinct optimization steps for the Discriminator and the Encoder-Decoder pair, managed by separate Adam optimizers. Crucially, during the training of one component, the parameters of the other components are frozen—a standard practice in adversarial training to ensure focused gradient updates.

The Discriminator is trained to maximize its ability to distinguish between latent vectors produced by the Encoder and those sampled directly from a multivariate normal distribution, which serves as the prior distribution for the WAE-GAN. The loss function for the Discriminator is defined as:

$$L_D = -\frac{1}{N}\sum_{i=1}^{N}\Big(\log\big(D(\mathbf{z}_{\text{random},i})\big) + \log\big(1 - D(\mathbf{z}_{\text{real},i})\big)\Big)$$

where $D(\cdot)$ is the Discriminator's output (probability of being real), z_{random} are samples from the prior, and z_{real} are latent vectors from the Encoder. This formulation, a non-saturating GAN loss with sigmoid output, aims to maximize the probability of the discriminator correctly identifying both real and random latent samples. The discriminator WAE regularization term is applied as a weighting factor to this loss, allowing for fine-tuning

the emphasis on the adversarial component. Gradient clipping with a maximum norm of 1 is applied to the Discriminator's parameters after backpropagation to prevent exploding gradients, common in deep neural networks and particularly beneficial in adversarial settings.

The Encoder and Decoder are trained jointly with a multifaceted loss function designed to serve two primary objectives: minimizing RE and encouraging the Encoder's latent space distribution to align with the prior distribution (by "fooling" the Discriminator). The combined loss for the Encoder-Decoder is:

$$L_{ED} = \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{L}_{\text{MSE}}(\hat{\mathbf{x}}_i, \mathbf{x}_i) + \mathcal{L}_{\text{CE}}(\hat{\mathbf{c}}_i, \mathbf{c}_i) - \text{WAE\_regularization\_term} \times \log(D(\mathbf{z}_{\text{Encoder},i}))\right)$$

Here, $\mathcal{L}_{MSE}$ is the MSE between the reconstructed input and the original data for continuous features. $\mathcal{L}_{CE}$ is the Cross-Entropy loss for reconstructed chambers against the true chambers labels. The term $\log(D(\mathbf{z}_{\text{Encoder}})$ is the Discriminator's output for the latent vector generated by the Encoder. By subtracting this term (which the Encoder-Decoder seeks to maximize), the Encoder is incentivized to produce latent representations that the Discriminator classifies as "real", thereby aligning its distribution with the prior. The Encoder-Decoder WAE regularization term acts as a weighting factor for this adversarial component.

A crucial observation during empirical testing was the need to adjust these WAE regularization term values. Specifically, for the Encoder/Decoder lower (e.g., 0.2) and for the Discriminator higher (e.g., 0.7) proved beneficial. This asymmetric weighting strategy is vital when the Discriminator becomes overly powerful too early in training. By reducing the adversarial pressure on the Encoder/Decoder, it allows the autoencoder part to prioritize learning effective data reconstruction, which is fundamental for AD, before focusing heavily on matching the latent space distribution. This helps to prevent issues where the autoencoder might sacrifice reconstruction quality to satisfy the Discriminator. Gradient clipping is also applied to the Encoder's and Decoder's parameters to manage gradient stability.

The Adam optimizer is used for all components. The Discriminator's learning rate is set to $1e^{-4}$, while the Encoder and Decoder share a learning rate of $1e^{-3}$. Each model is trained with 30 epochs. Training and validation losses are monitored.

A dedicated part is used after each epoch (or block of training steps) to assess the model's

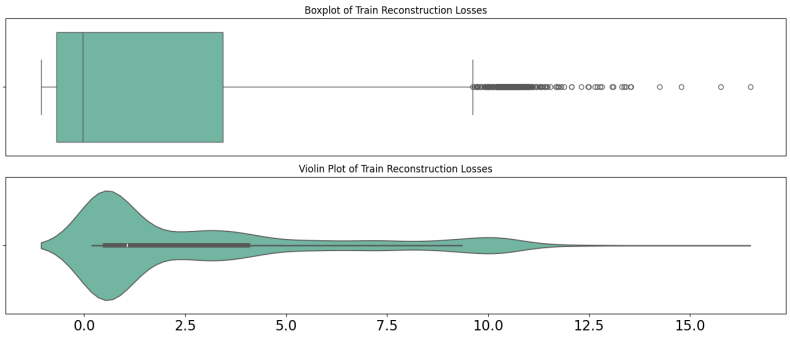
generalization capabilities on unseen data. During validation, the Encoder and Decoder are set to evaluation mode and gradient computations are disabled. The validation loss is computed as the sum of MSE loss for continuous features and Cross-Entropy loss for chamber features. Monitoring this validation loss is critical to detect overfitting, ensuring that the model learns generalizable patterns from the training data rather than merely memorizing it. A consistently low validation loss relative to training loss indicates robust performance, while a significant divergence might signal overfitting, prompting further hyperparameter tuning or regularization adjustments.

Regarding the AD threshold is determined based on the statistical distribution of train REs—the same way as the LSTM-AE model, explained in the last paragraph of B.1.

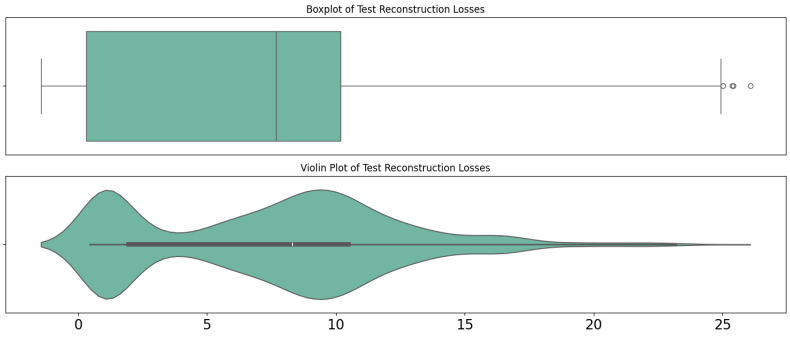
C. Supplemental Visualizations

C.1. LSTM-AE Models

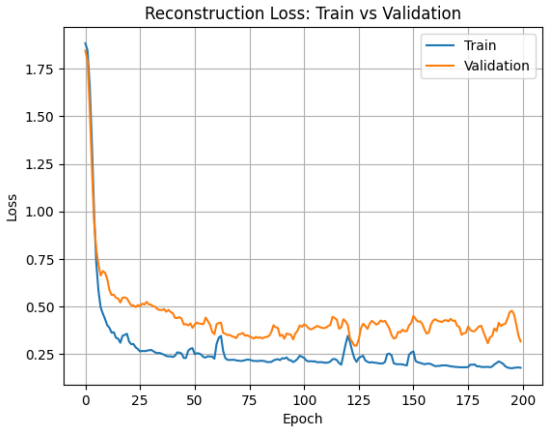
The following Figures present additional visualizations for the LSTM-AE models for each PUnit, including their reconstruction loss curves, reconstruction error distributions, and confusion matrix.



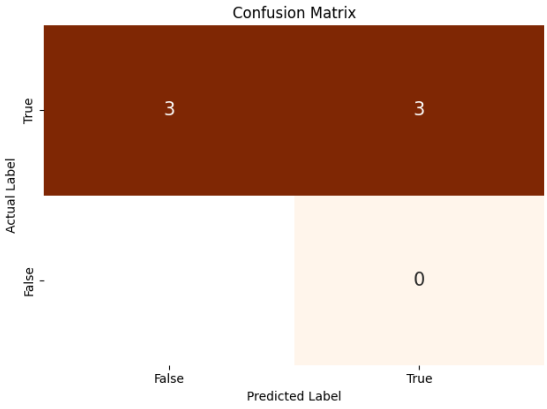
(a) Reconstruction Train Errors Distribution



(b) Reconstruction Test Errors Distribution

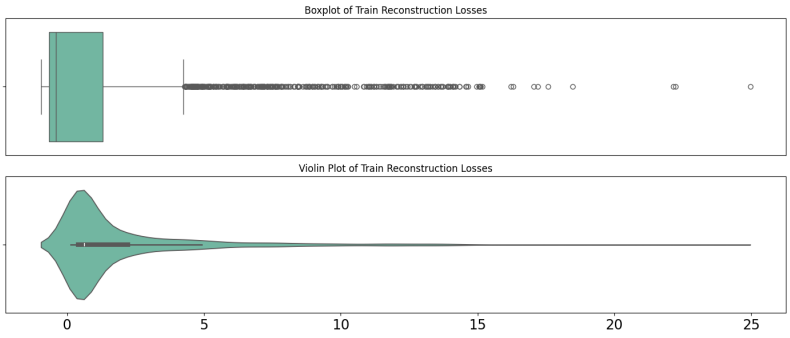


(c) Reconstruction Loss

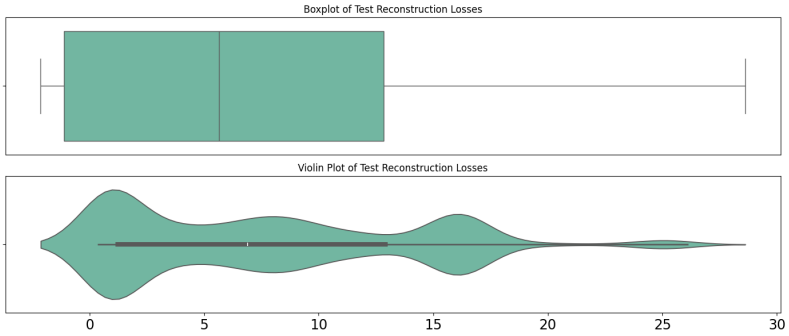


(d) Confusion Matrix

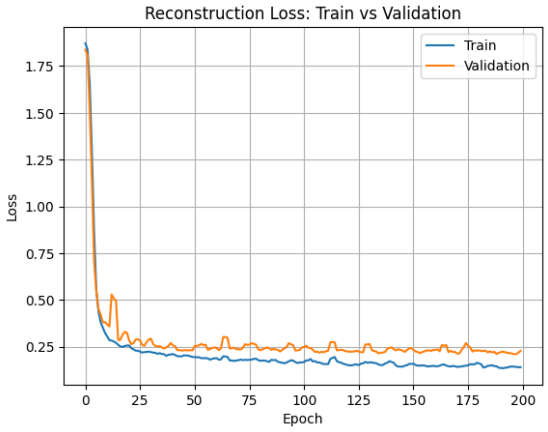
Figure C.1: Supplemental Visualizations for `lstm-ae-punit3-1` - PUnit 3



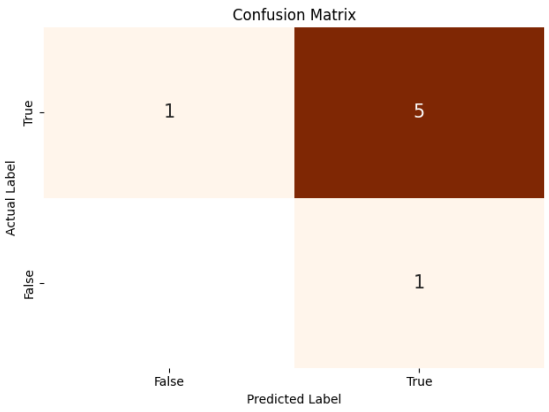
(a) Reconstruction Train Errors Distribution



(b) Reconstruction Test Errors Distribution



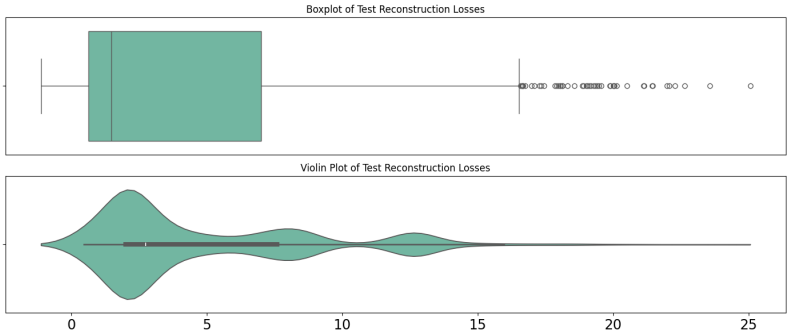
(c) Reconstruction Loss



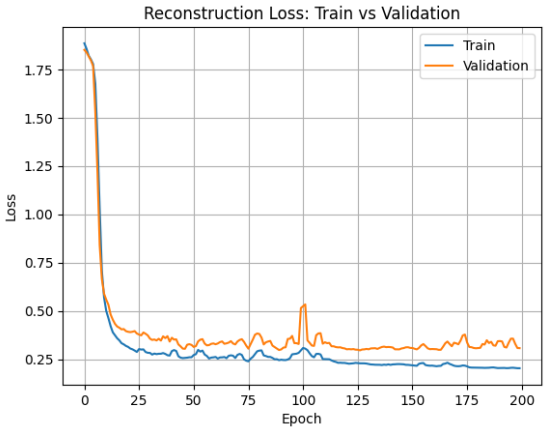
(d) Confusion Matrix

Figure C.2: Supplemental Visualizations for `lstm-ae-punit3-2` - PUnit 3

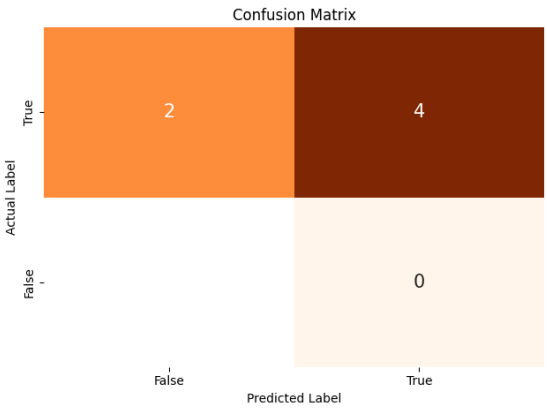
(a) Reconstruction Train Errors Distribution



(b) Reconstruction Test Errors Distribution

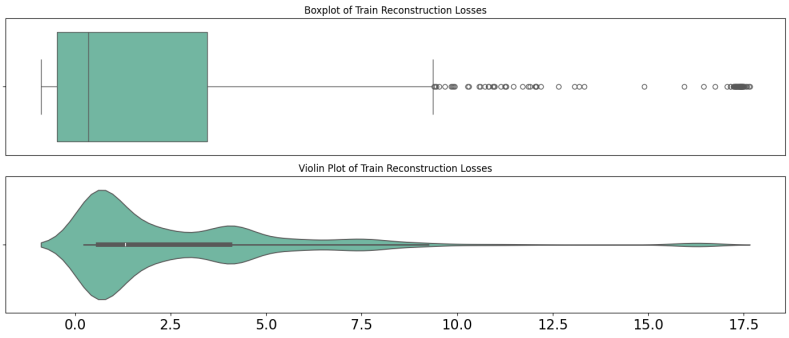


(c) Reconstruction Loss



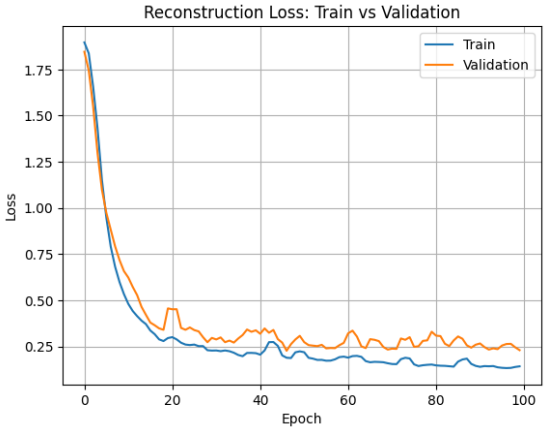
(d) Confusion Matrix

Figure C.3: Supplemental Visualizations for `lstm-ae-punit3-3` - PUnit 3

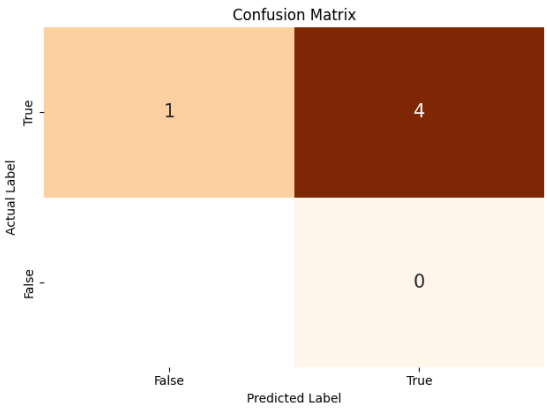


(a) Reconstruction Train Errors Distribution

(b) Reconstruction Test Errors Distribution

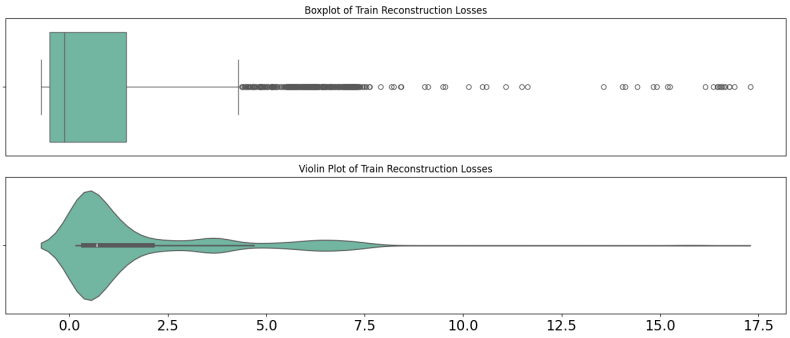


(c) Reconstruction Loss

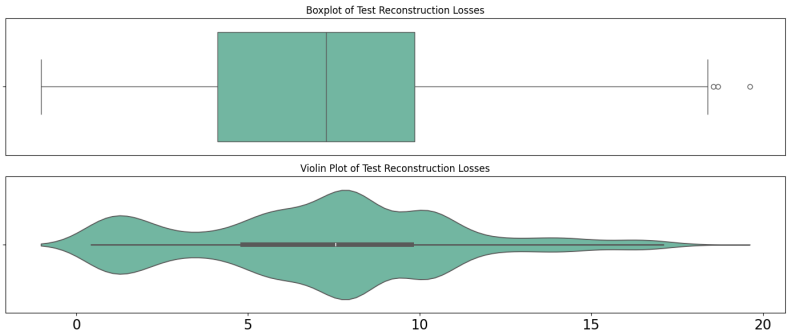


(d) Confusion Matrix

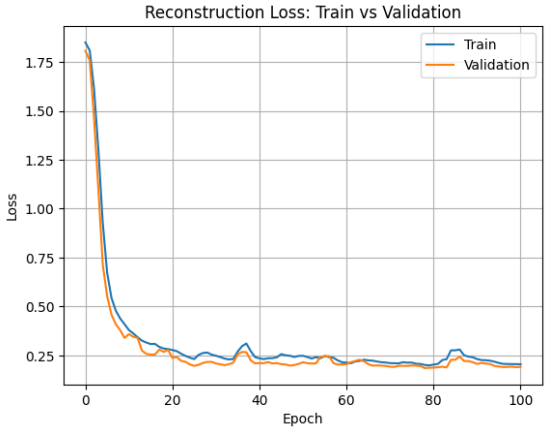
Figure C.4: Supplemental Visualizations for `lstm-ae-punit10-1` - PUnit 10



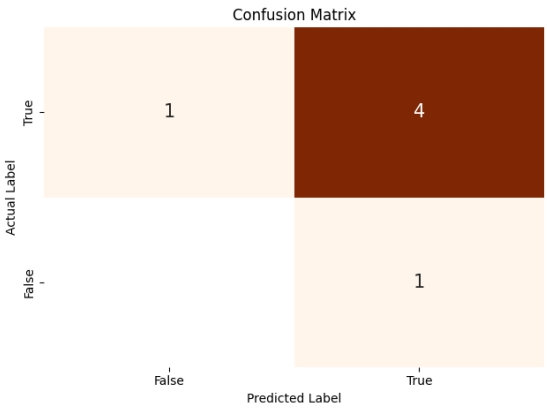
(a) Reconstruction Train Errors Distribution



(b) Reconstruction Test Errors Distribution



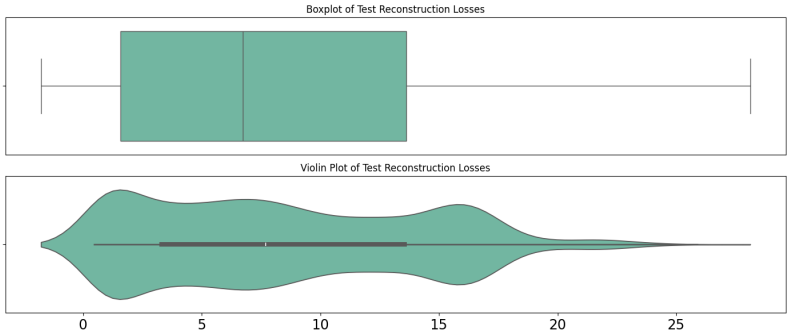
(c) Reconstruction Loss



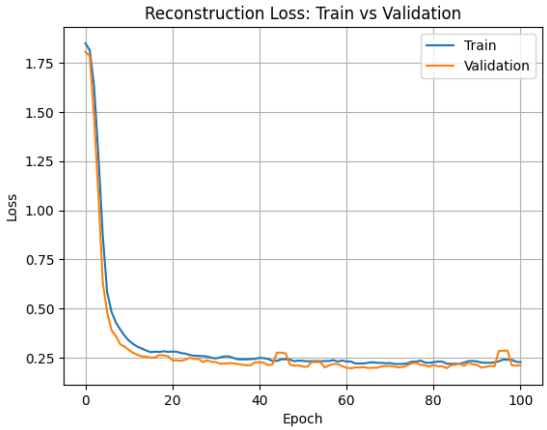
(d) Confusion Matrix

Figure C.5: Supplemental Visualizations for `lstm-ae-punit10-2` - PUnit 10

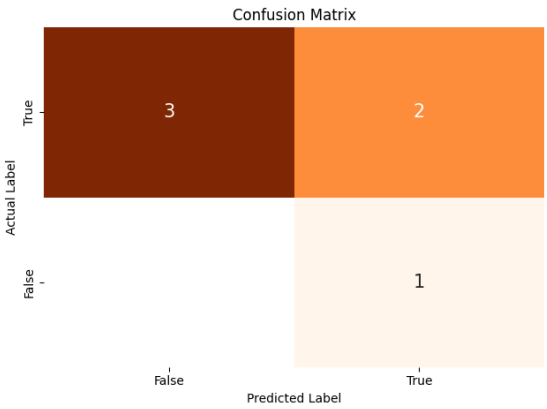
(a) Reconstruction Train Errors Distribution



(b) Reconstruction Test Errors Distribution

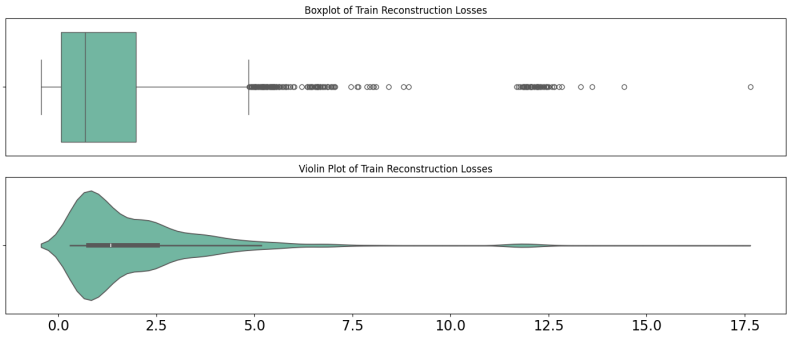


(c) Reconstruction Loss

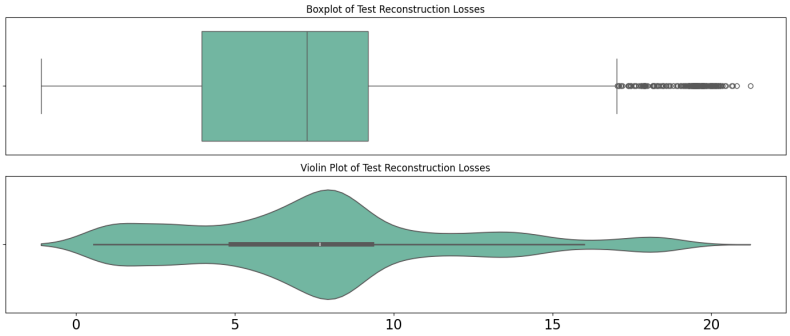


(d) Confusion Matrix

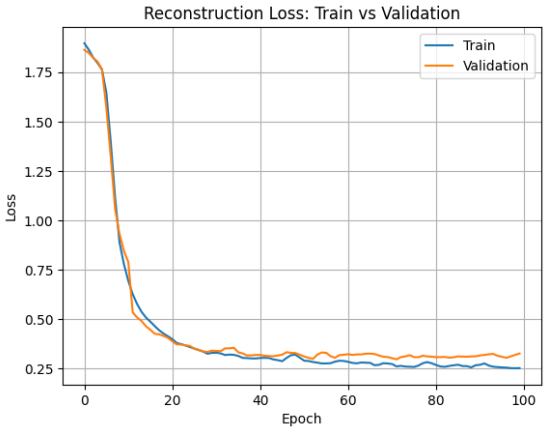
Figure C.6: Supplemental Visualizations for `lstm-ae-punit10-3` - PUnit 10



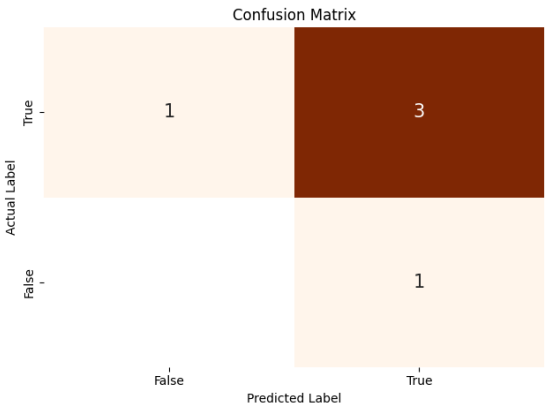
(a) Reconstruction Train Errors Distribution



(b) Reconstruction Test Errors Distribution

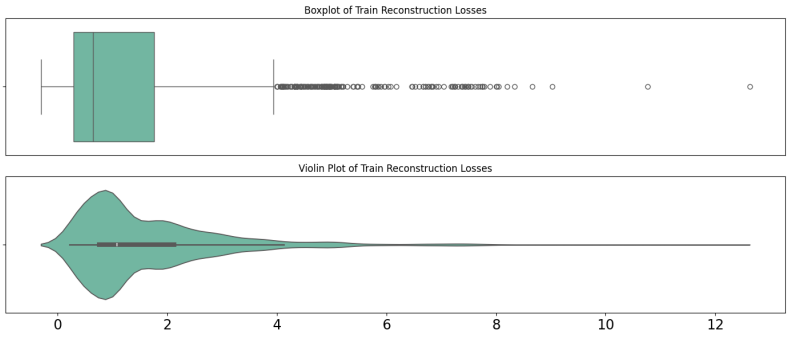


(c) Reconstruction Loss

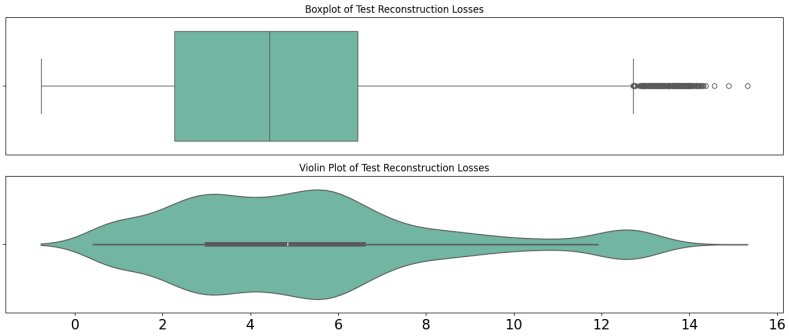


(d) Confusion Matrix

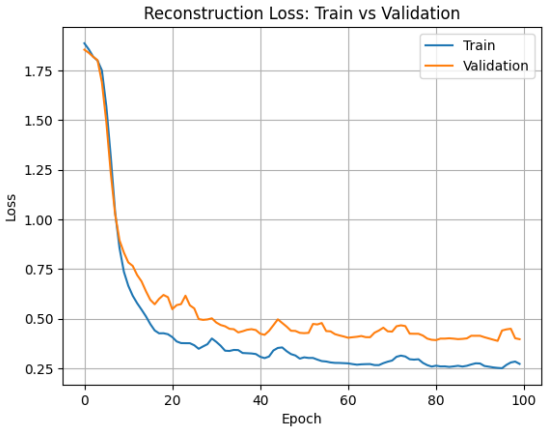
Figure C.7: Supplemental Visualizations for `lstm-ae-punit12-1` - PUnit 12



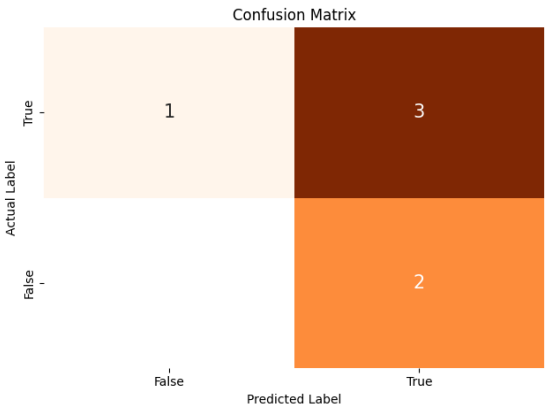
(a) Reconstruction Train Errors Distribution



(b) Reconstruction Test Errors Distribution

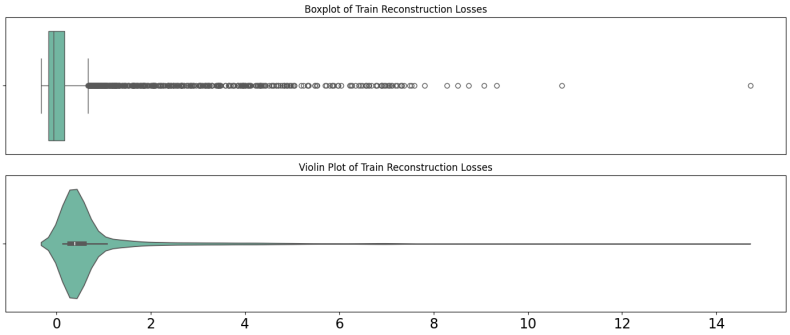


(c) Reconstruction Loss

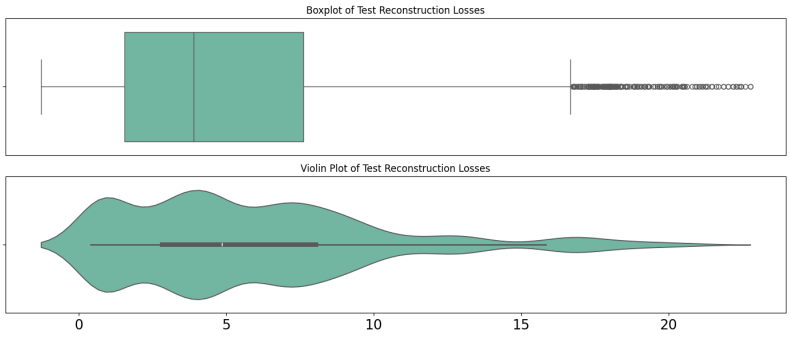


(d) Confusion Matrix

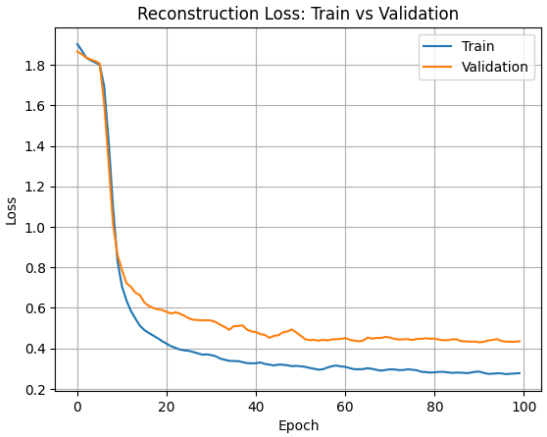
Figure C.8: Supplemental Visualizations for `lstm-ae-punit12-2` - PUnit 12



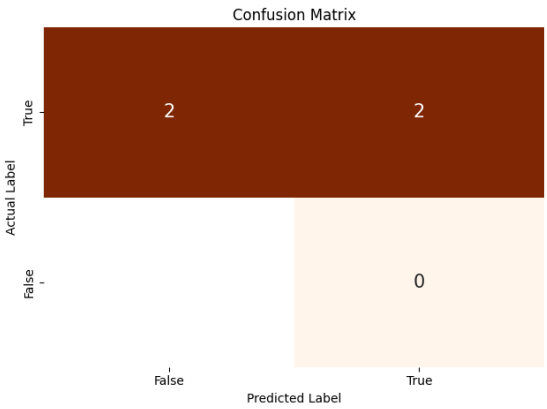
(a) Reconstruction Train Errors Distribution



(b) Reconstruction Test Errors Distribution



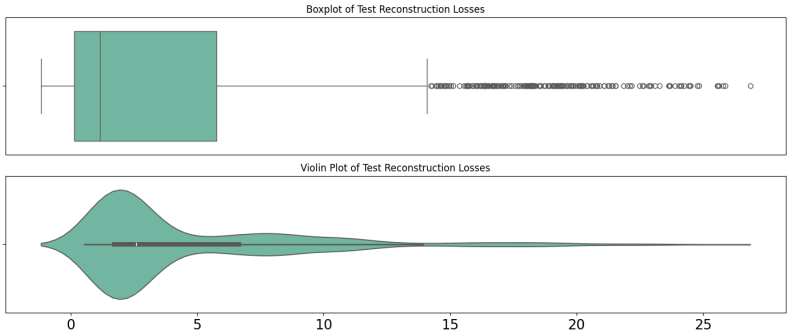
(c) Reconstruction Loss



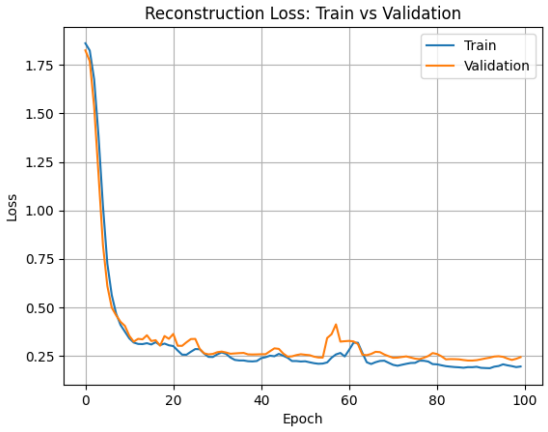
(d) Confusion Matrix

Figure C.9: Supplemental Visualizations for `lstm-ae-punit12-3` - PUnit 12

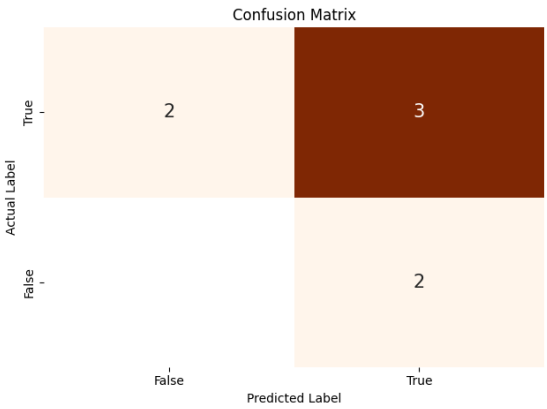
(a) Reconstruction Train Errors Distribution



(b) Reconstruction Test Errors Distribution

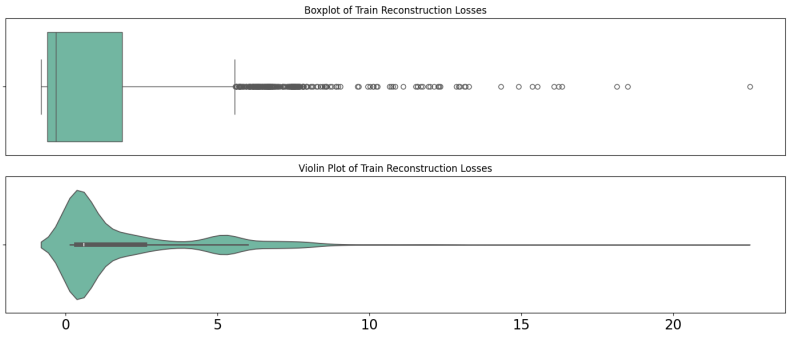


(c) Reconstruction Loss

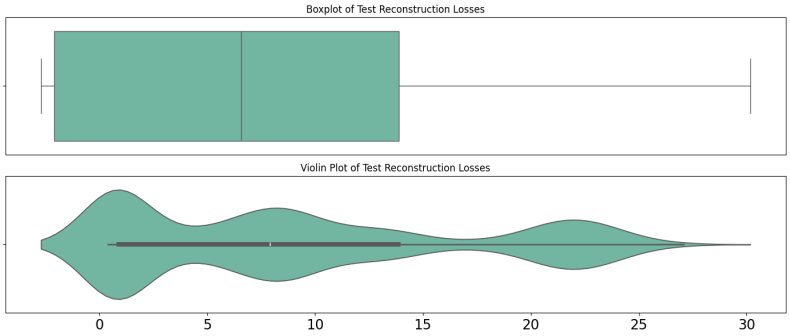


(d) Confusion Matrix

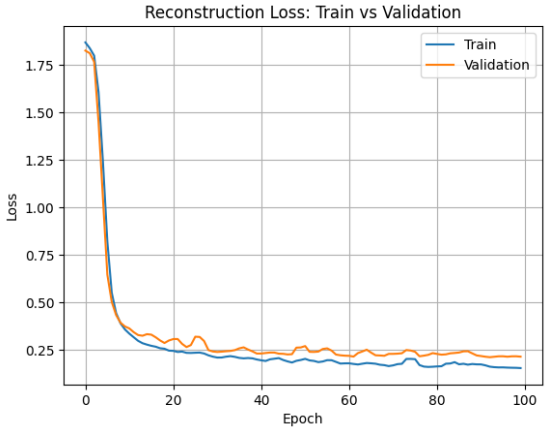
Figure C.10: Supplemental Visualizations for `lstm-ae-punit15-1` - PUnit 15



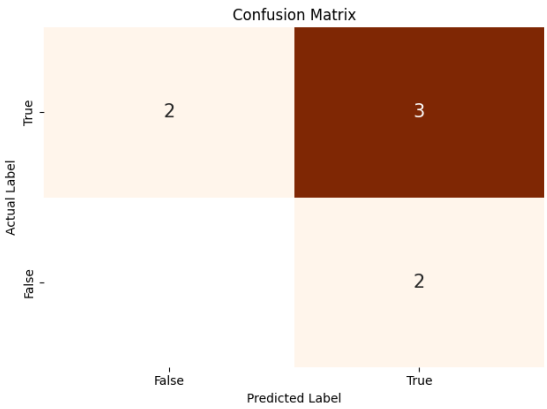
(a) Reconstruction Train Errors Distribution



(b) Reconstruction Test Errors Distribution

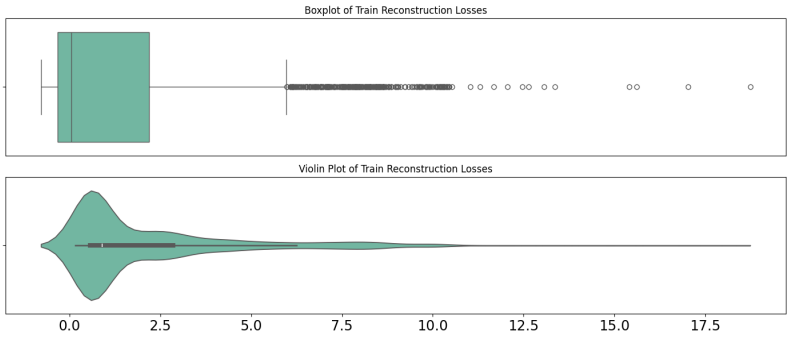


(c) Reconstruction Loss



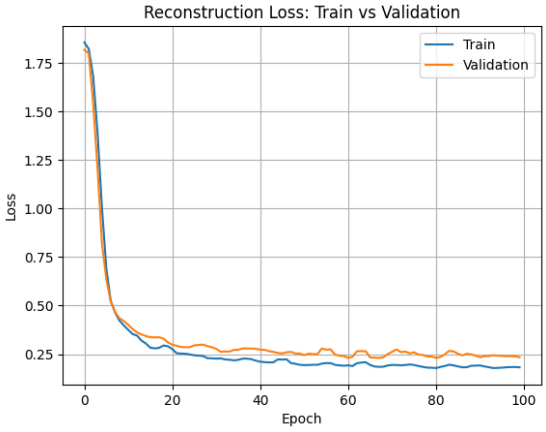
(d) Confusion Matrix

Figure C.11: Supplemental Visualizations for `lstm-ae-punit15-2` - PUnit 15

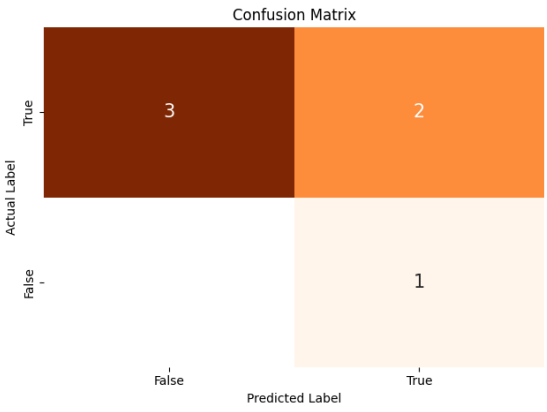


(a) Reconstruction Train Errors Distribution

(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

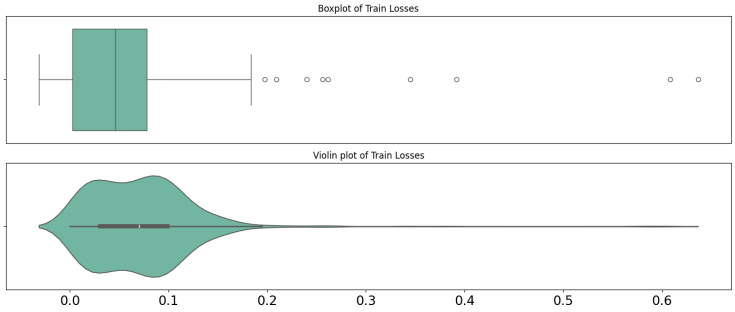


(d) Confusion Matrix

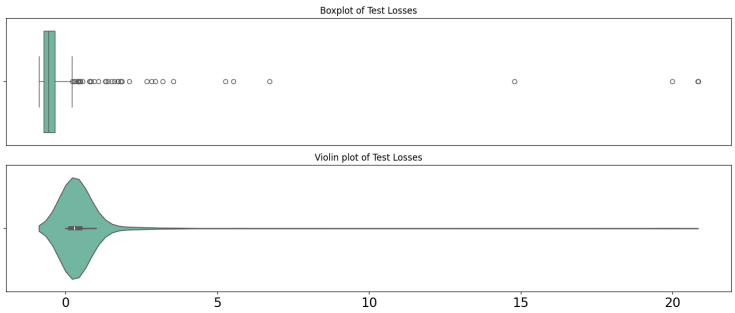
Figure C.12: Supplemental Visualizations for `lstm-ae-punit15-3` - PUnit 15

C.2. WAE-GAN Models

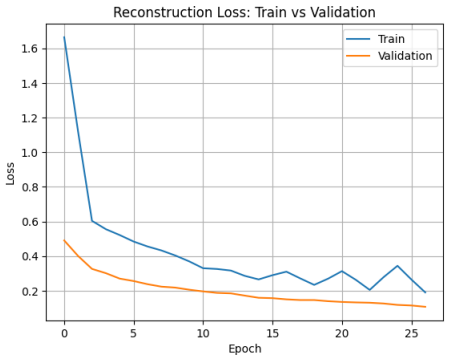
The following Figures present additional visualizations for the WAE-GAN models for each PUnit, including their reconstruction error distributions, reconstruction loss curves, and confusion matrices.



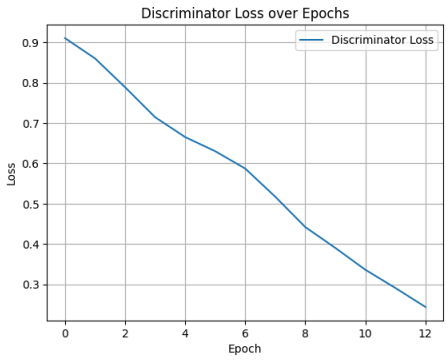
(a) Reconstruction Train Errors Distribution



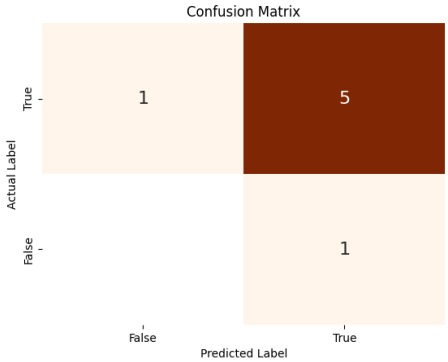
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

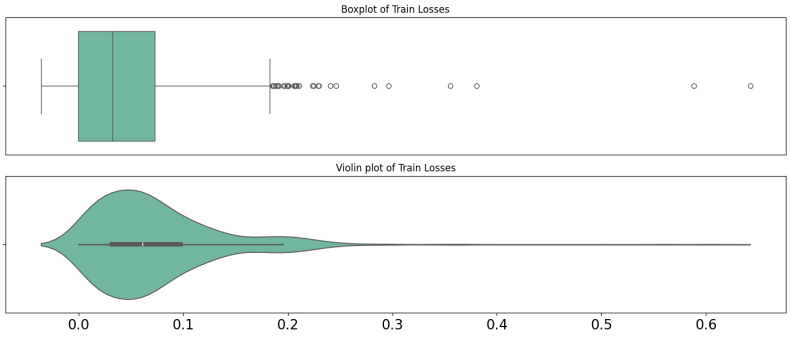


(d) Discriminator Loss

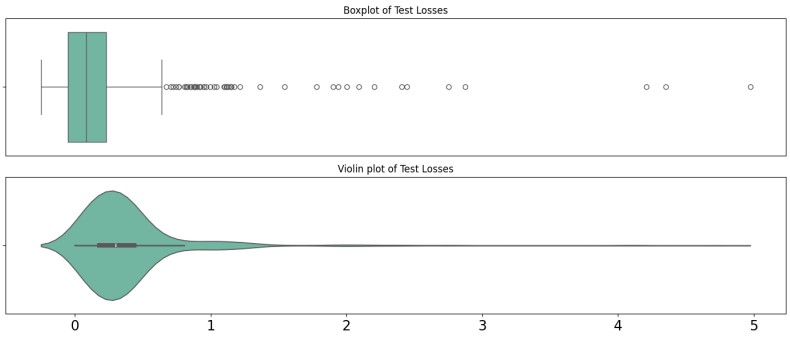


(e) Confusion Matrix

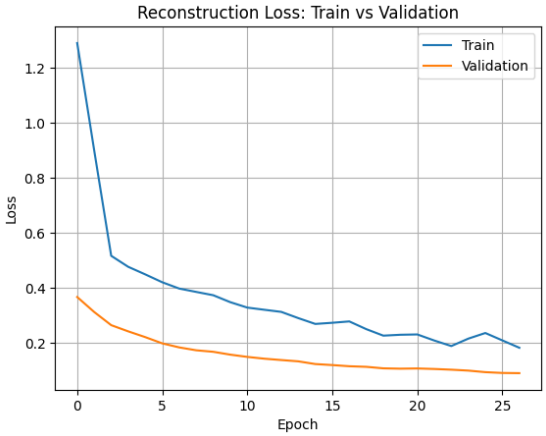
Figure C.13: Supplemental Visualizations for `wae-gan-punit3-1` - PUnit 3



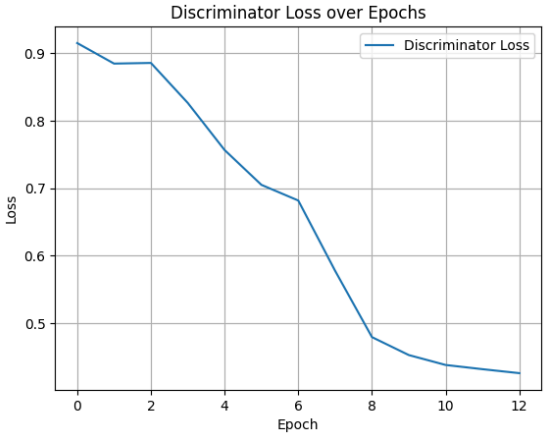
(a) Reconstruction Train Errors Distribution



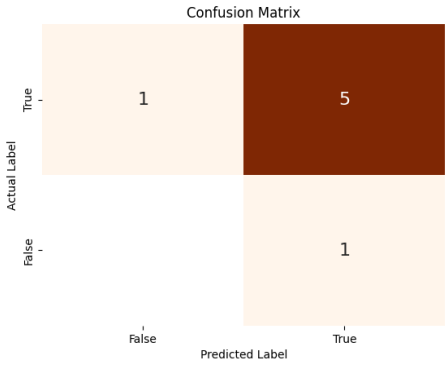
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

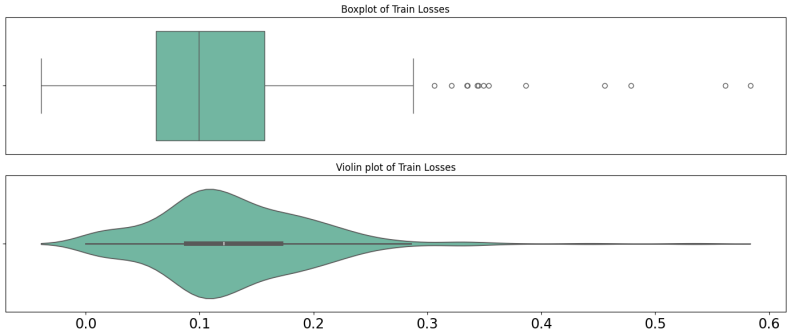


(d) Discriminator Loss

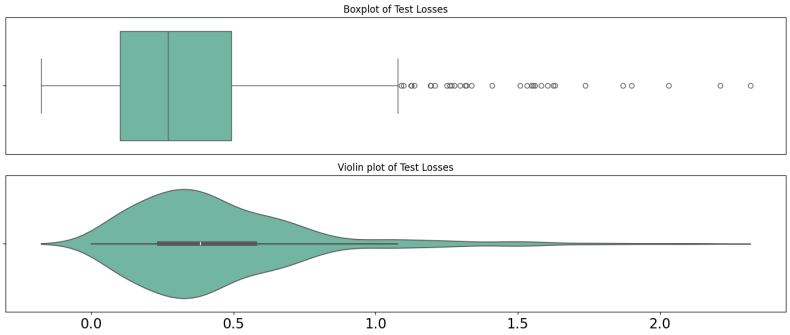


(e) Confusion Matrix

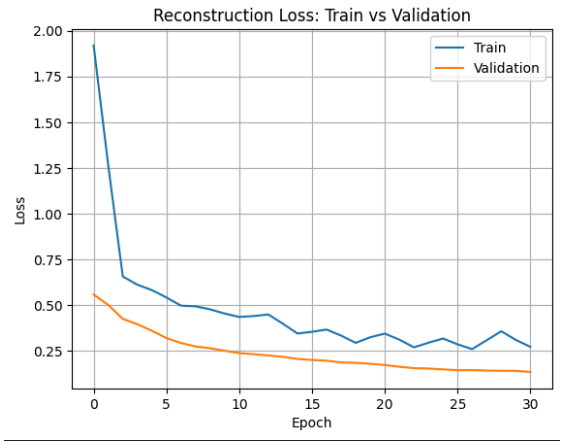
Figure C.14: Supplemental Visualizations for `wae-gan-punit3-2` - PUnit 3



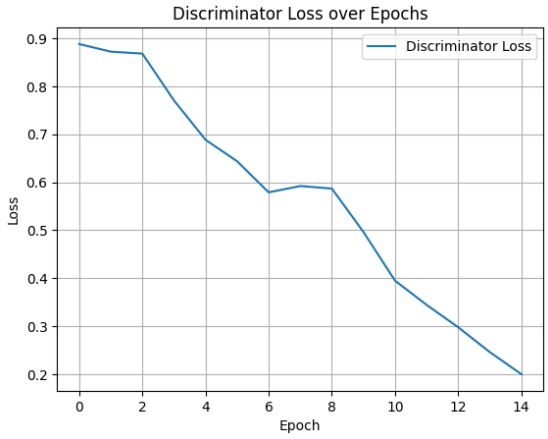
(a) Reconstruction Train Errors Distribution



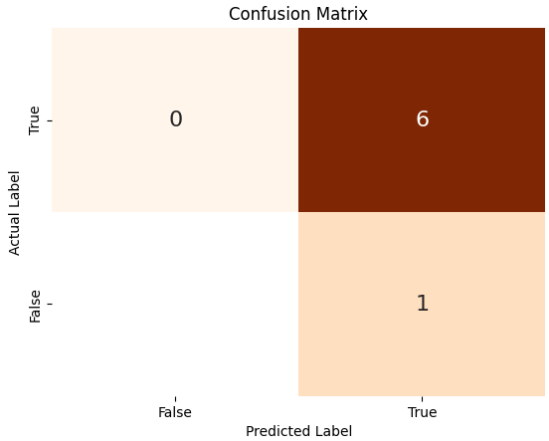
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

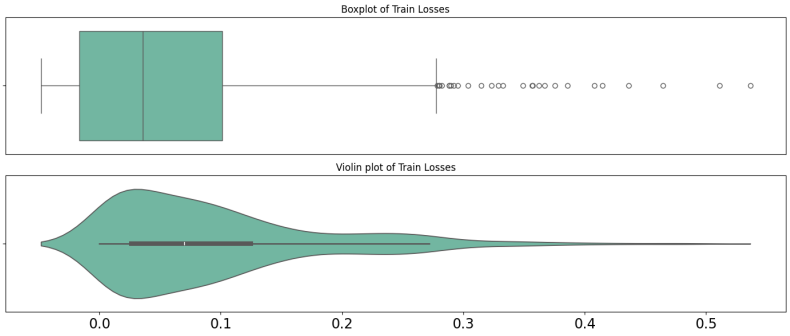


(d) Discriminator Loss

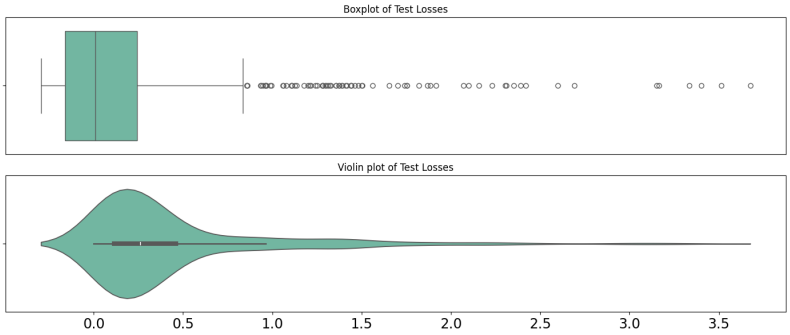


(e) Confusion Matrix

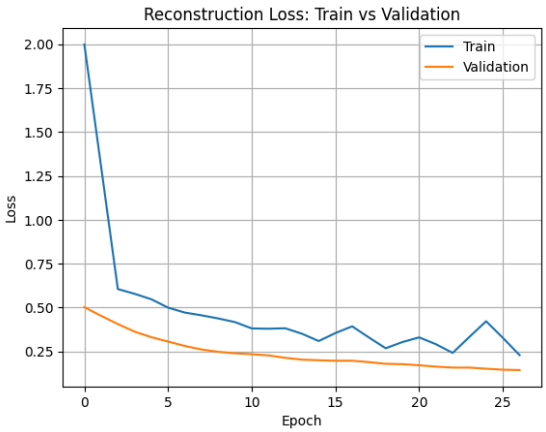
Figure C.15: Supplemental Visualizations for `wae-gan-punit3-3` - PUnit 3



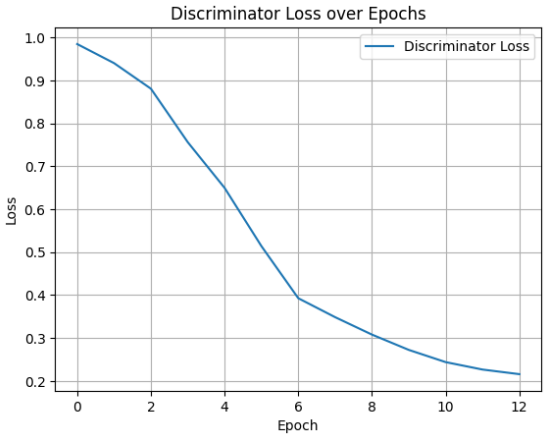
(a) Reconstruction Train Errors Distribution



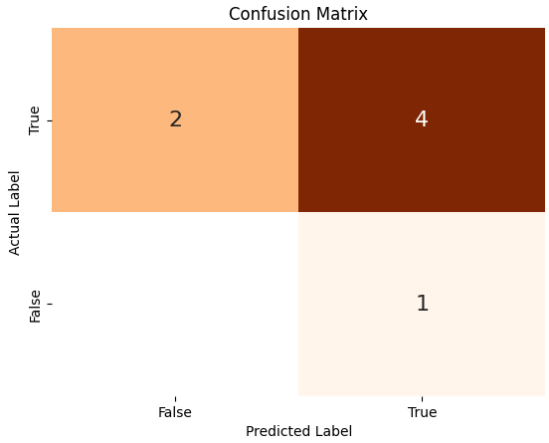
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

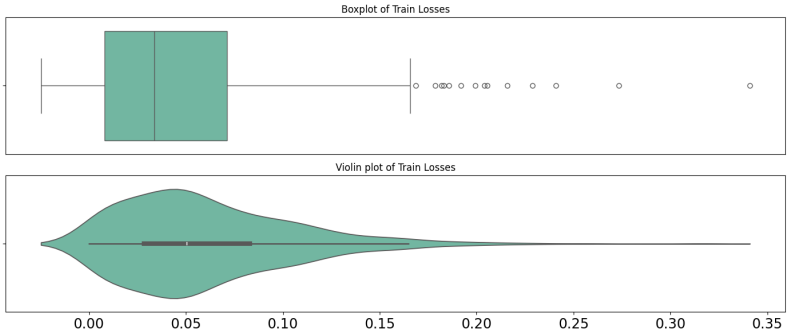


(d) Discriminator Loss

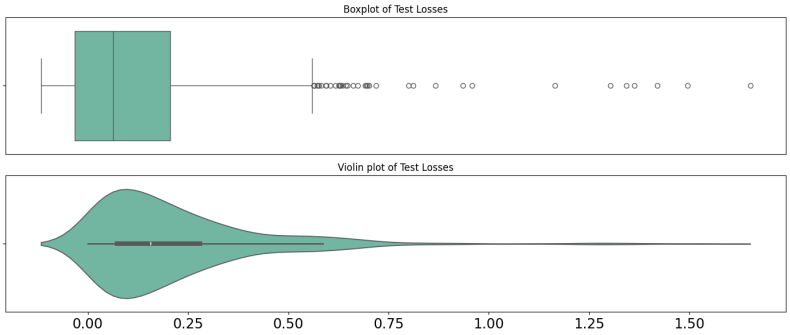


(e) Confusion Matrix

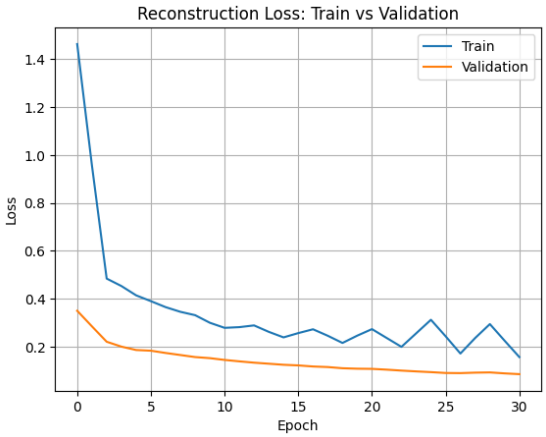
Figure C.16: Supplemental Visualizations for `wae-gan-punit3-4` - PUnit 3



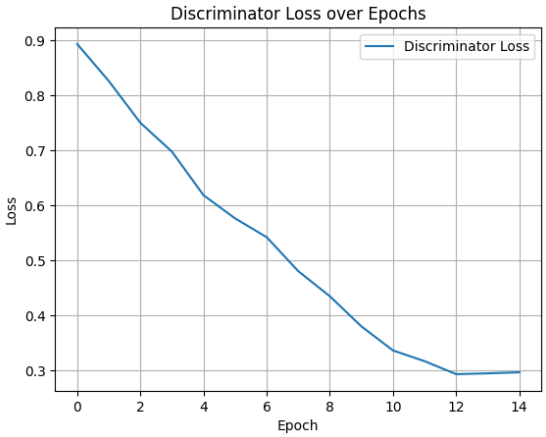
(a) Reconstruction Train Errors Distribution



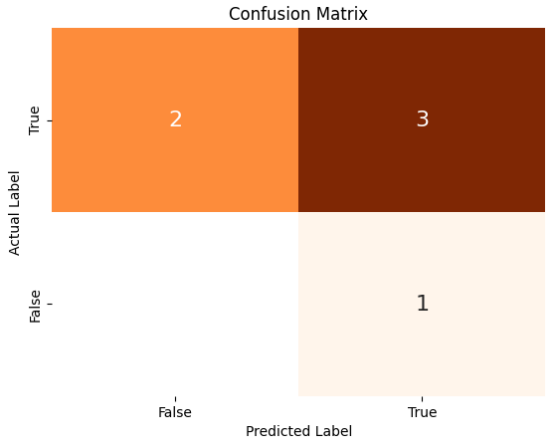
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

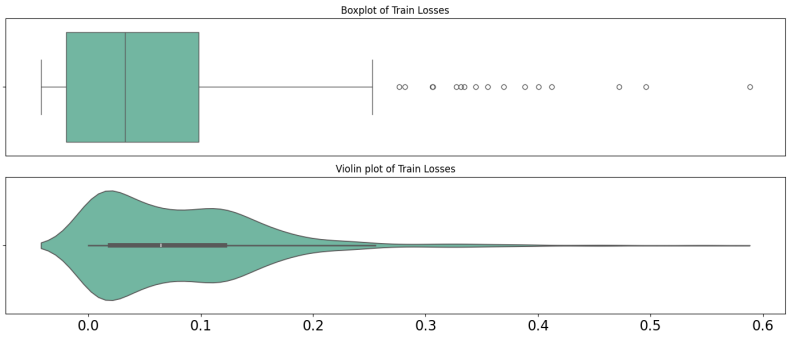


(d) Discriminator Loss

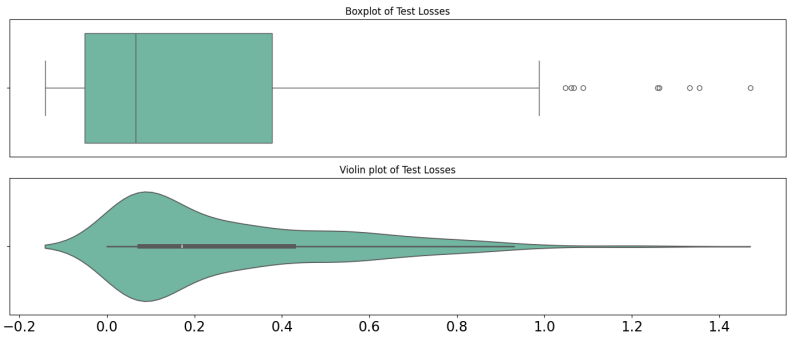


(e) Confusion Matrix

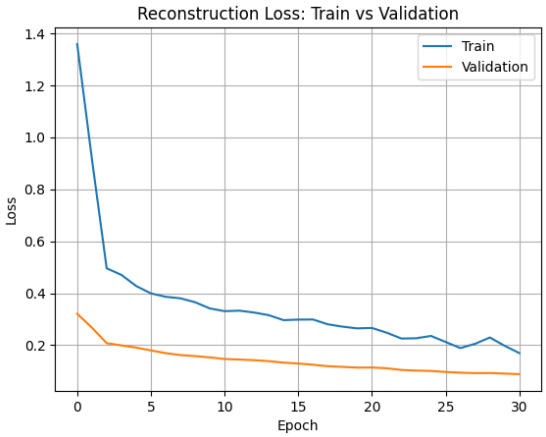
Figure C.17: Supplemental Visualizations for `wae-gan-punit10-1` - PUnit 10



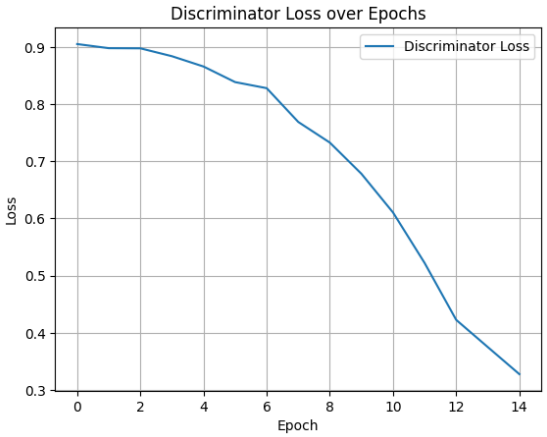
(a) Reconstruction Train Errors Distribution



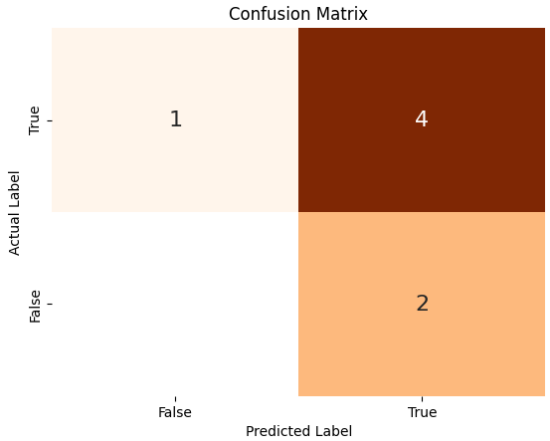
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

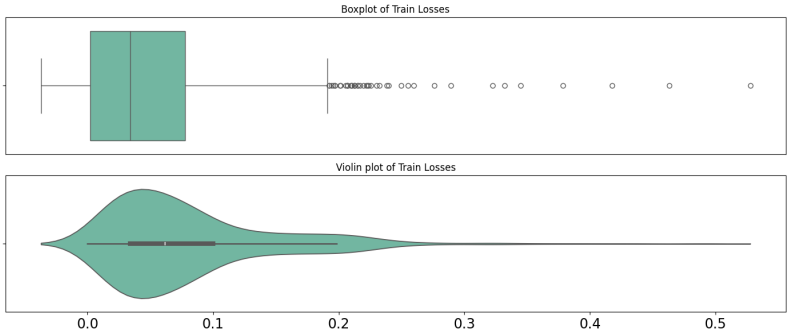


(d) Discriminator Loss

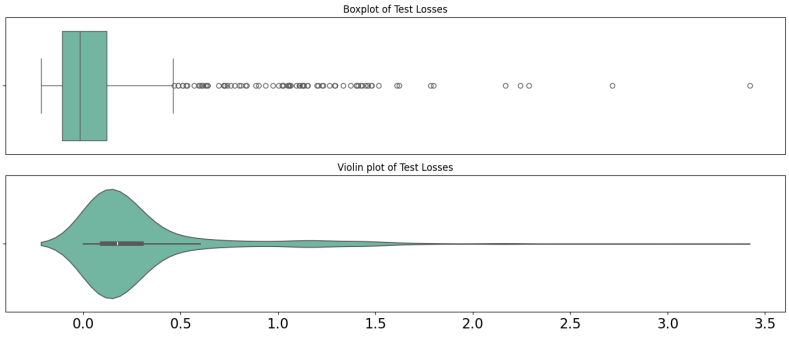


(e) Confusion Matrix

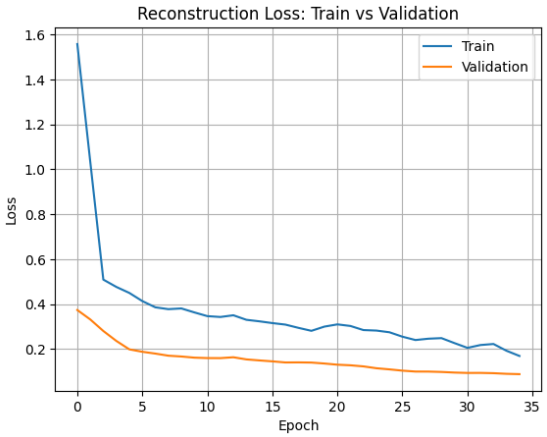
Figure C.18: Supplemental Visualizations for `wae-gan-punit10-2` - PUnit 10



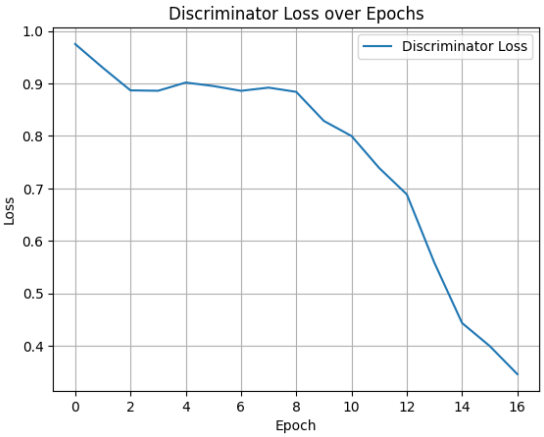
(a) Reconstruction Train Errors Distribution



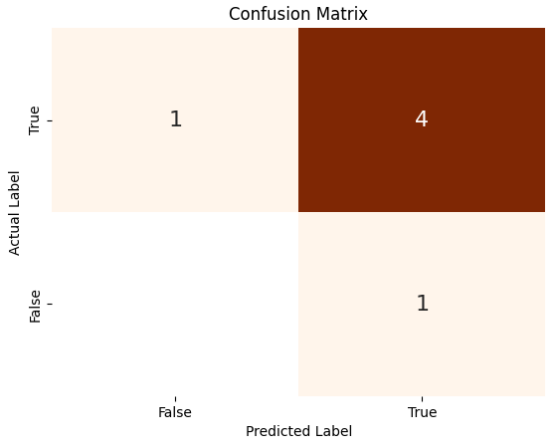
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

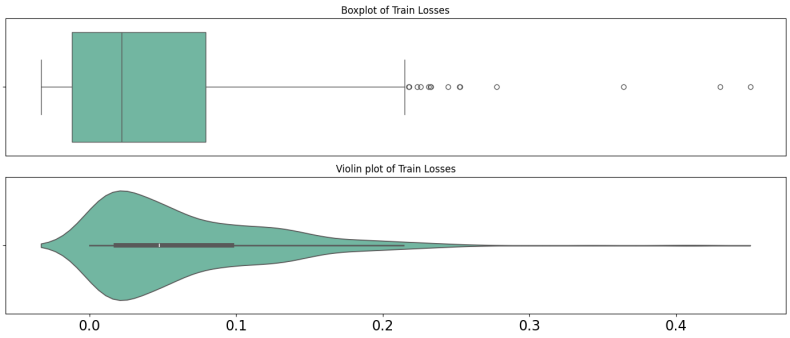


(d) Discriminator Loss

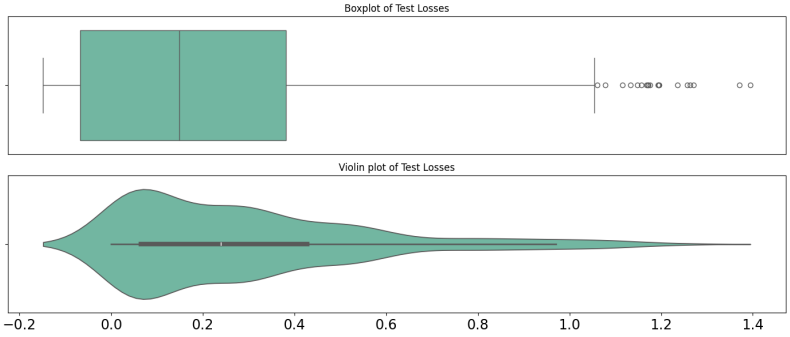


(e) Confusion Matrix

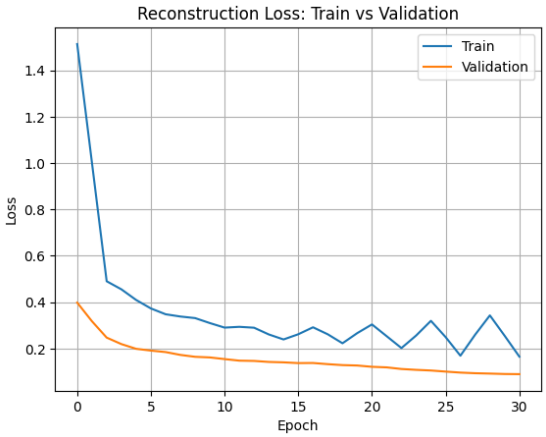
Figure C.19: Supplemental Visualizations for `wae-gan-punit10-3` - PUnit 10



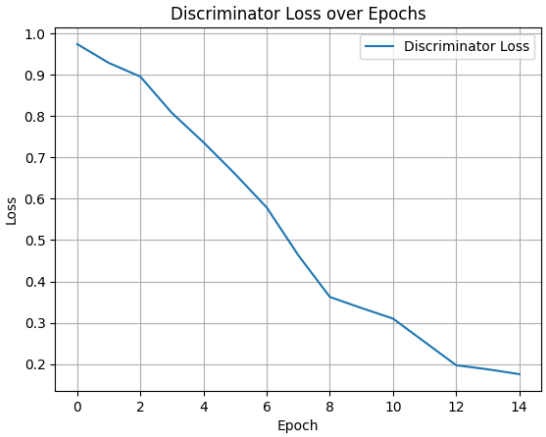
(a) Reconstruction Train Errors Distribution



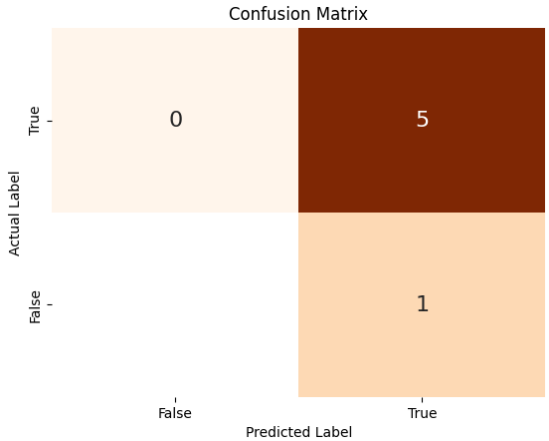
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

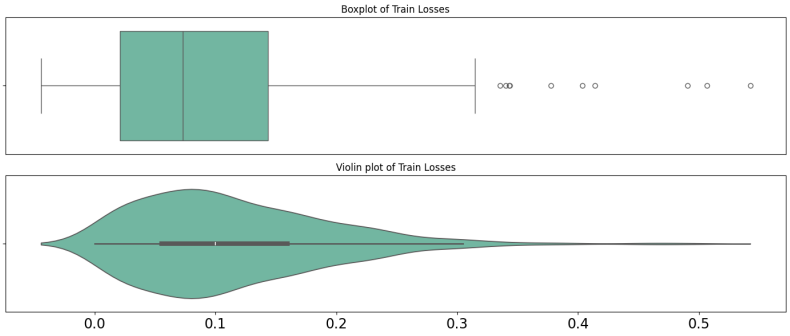


(d) Discriminator Loss

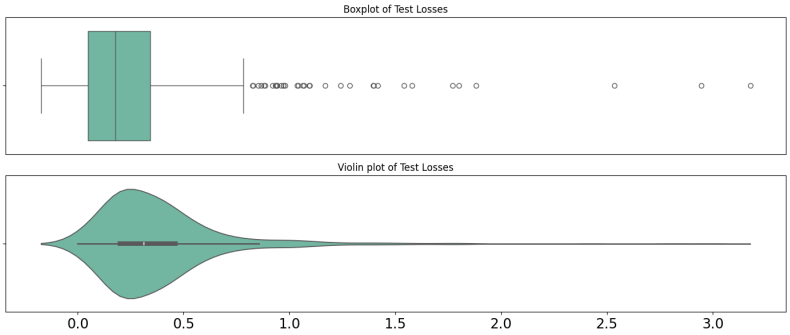


(e) Confusion Matrix

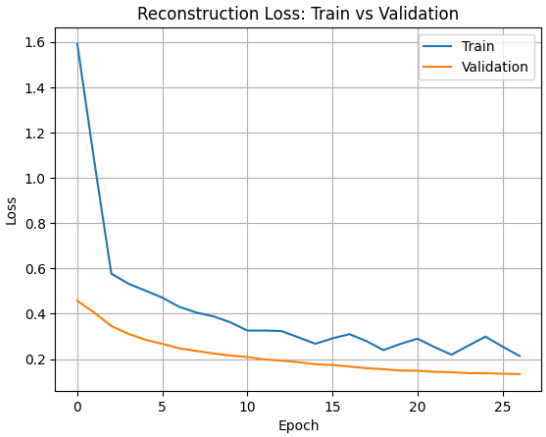
Figure C.20: Supplemental Visualizations for `wae-gan-punit10-4` - PUnit 10



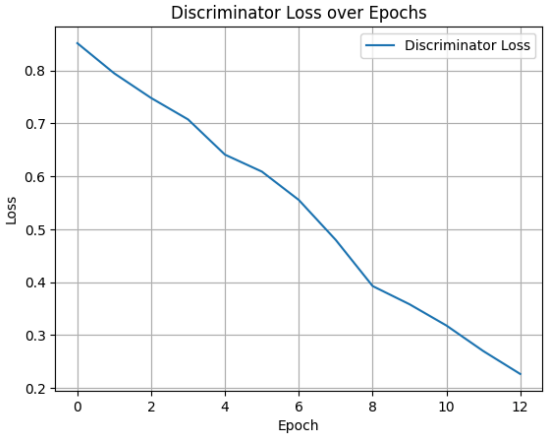
(a) Reconstruction Train Errors Distribution



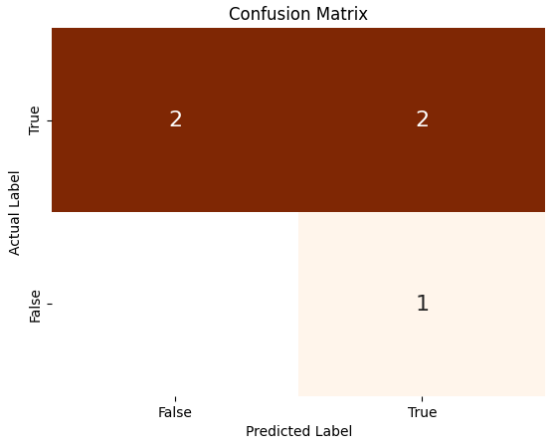
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

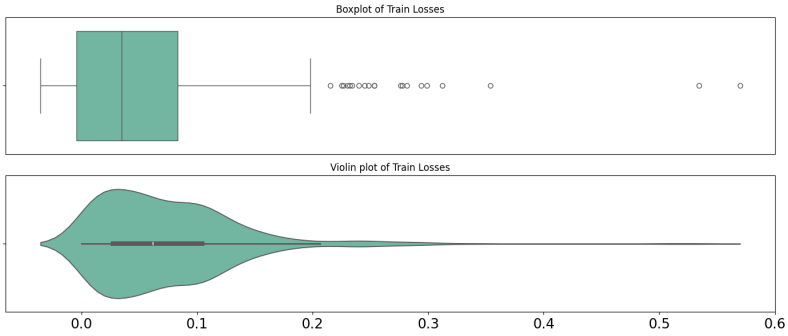


(d) Discriminator Loss

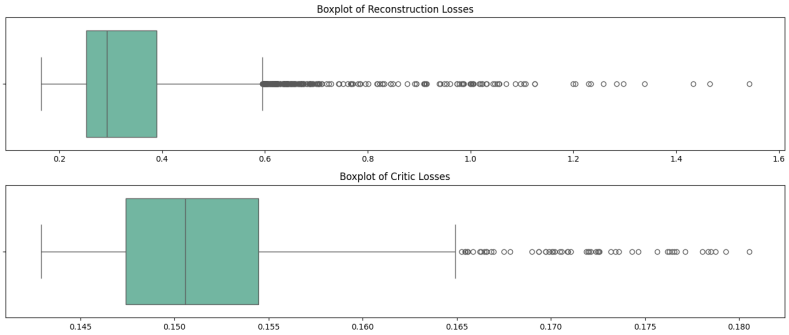


(e) Confusion Matrix

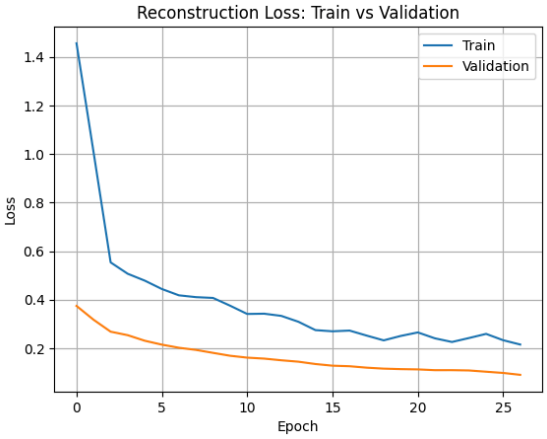
Figure C.21: Supplemental Visualizations for `wae-gan-punit12-1` - PUnit 12



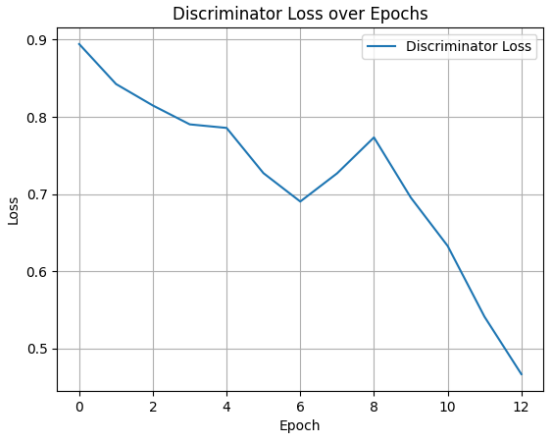
(a) Reconstruction Train Errors Distribution



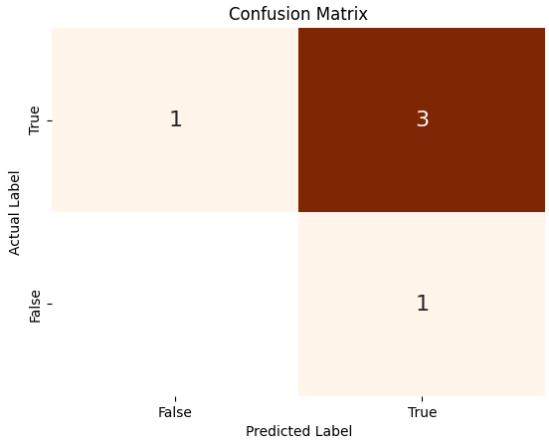
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

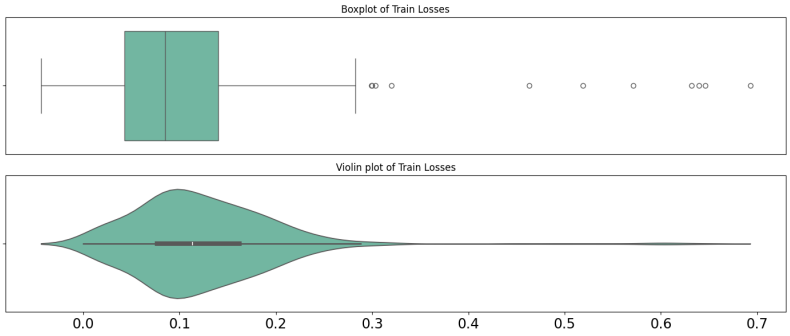


(d) Discriminator Loss

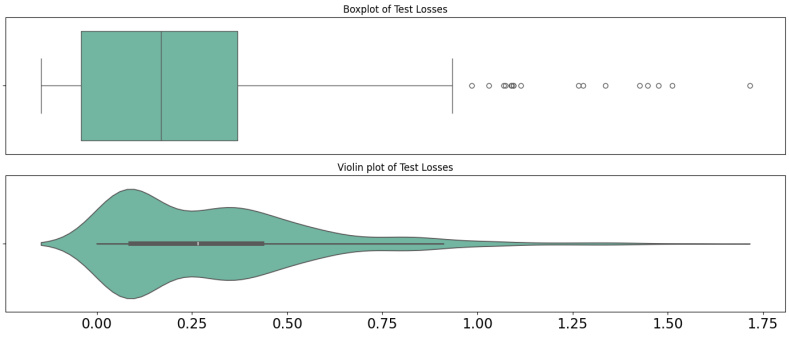


(e) Confusion Matrix

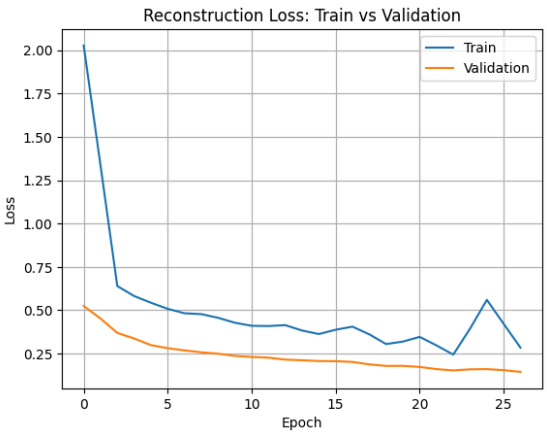
Figure C.22: Supplemental Visualizations for `wae-gan-punit12-2` - PUnit 12



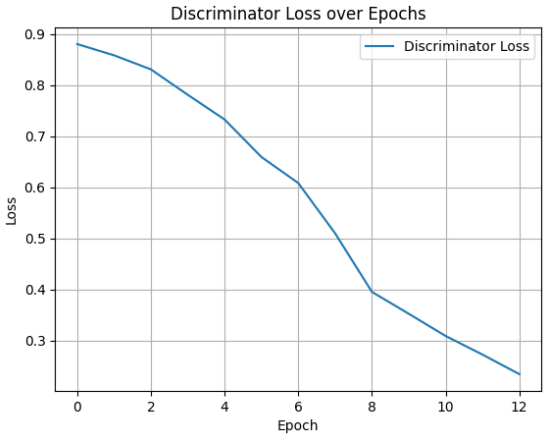
(a) Reconstruction Train Errors Distribution



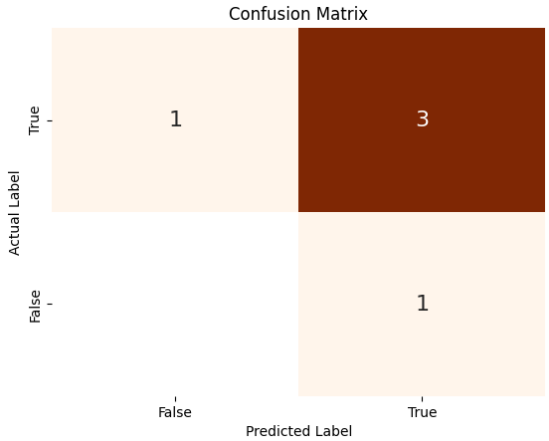
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

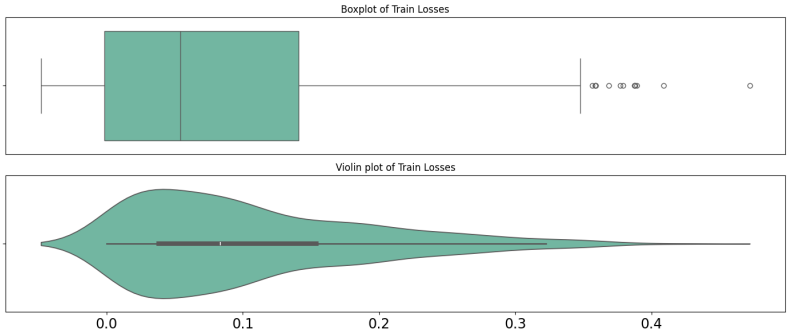


(d) Discriminator Loss

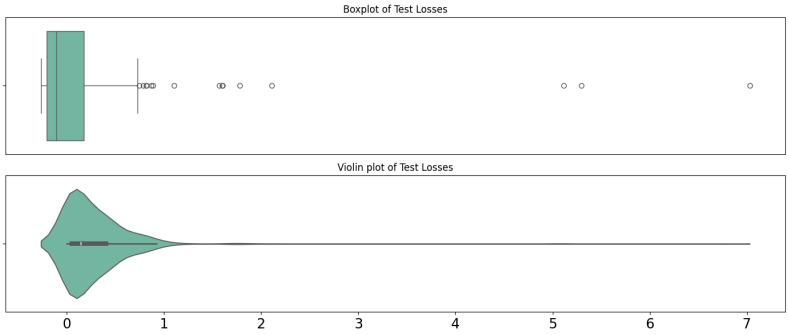


(e) Confusion Matrix

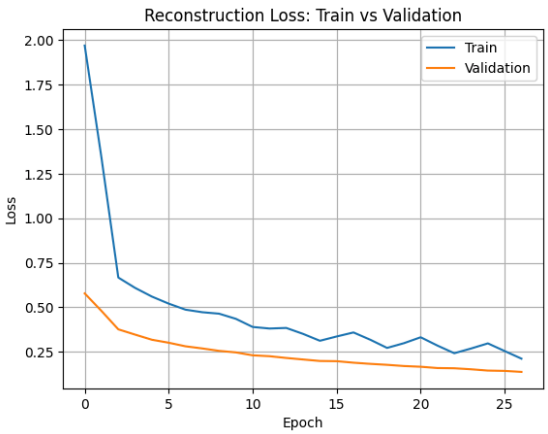
Figure C.23: Supplemental Visualizations for `wae-gan-punit12-3` - PUnit 12



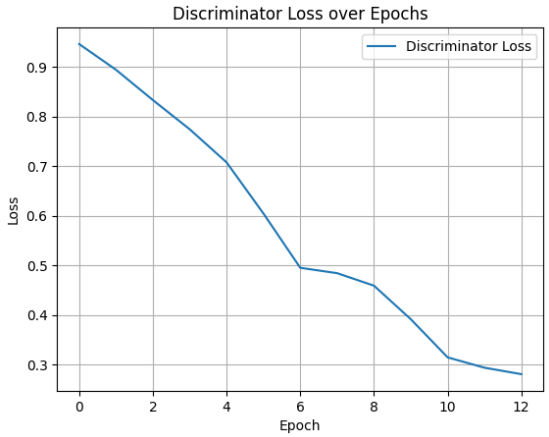
(a) Reconstruction Train Errors Distribution



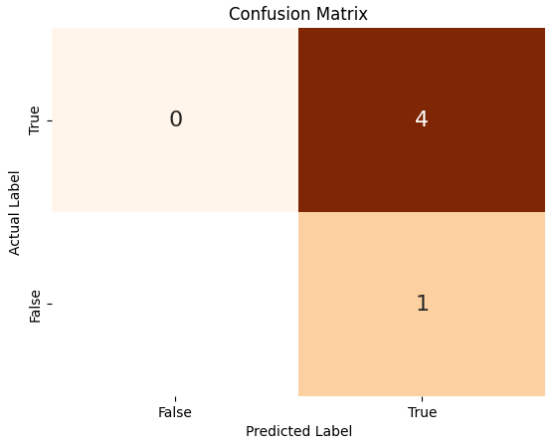
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

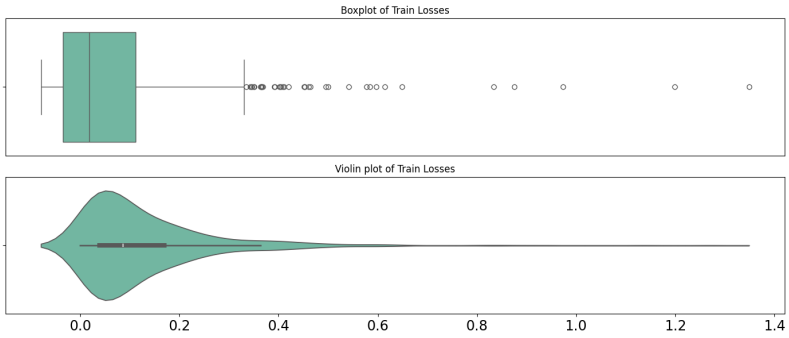


(d) Discriminator Loss

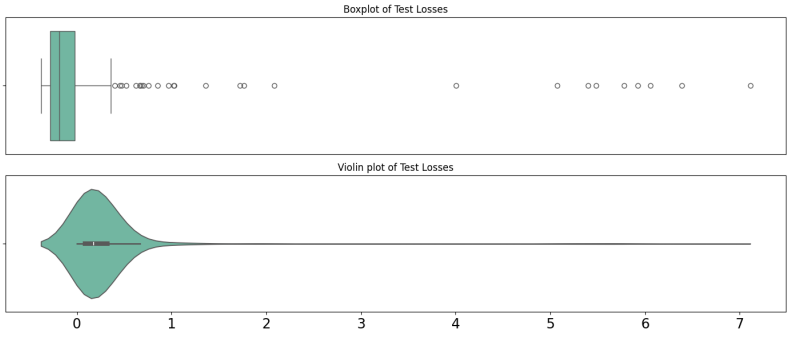


(e) Confusion Matrix

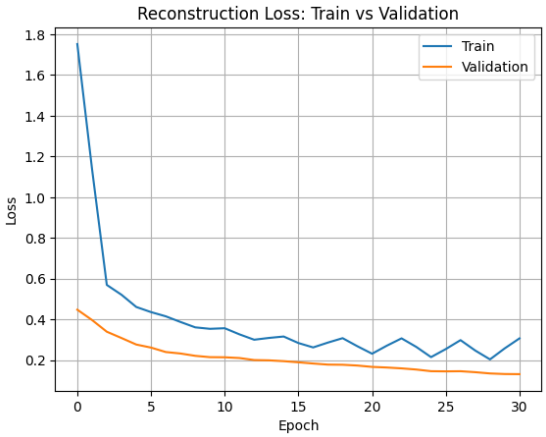
Figure C.24: Supplemental Visualizations for `wae-gan-punit12-4` - PUnit 12



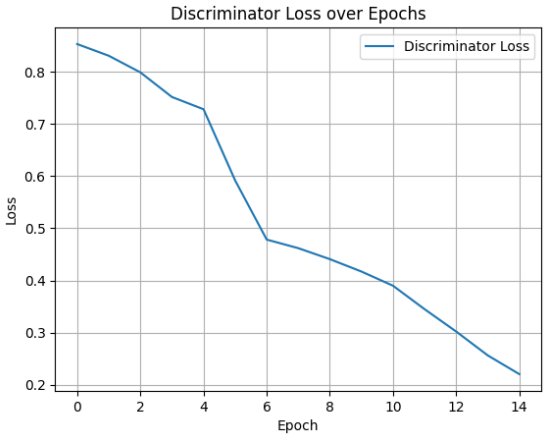
(a) Reconstruction Train Errors Distribution



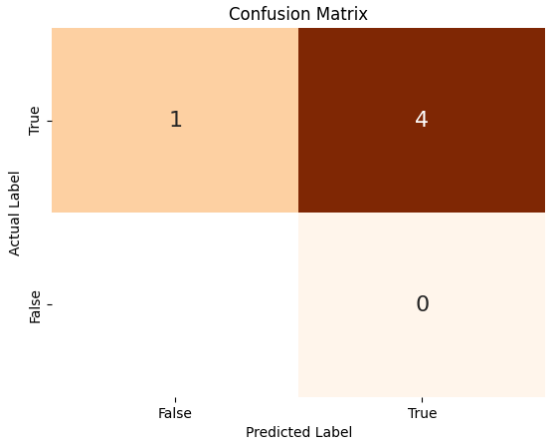
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

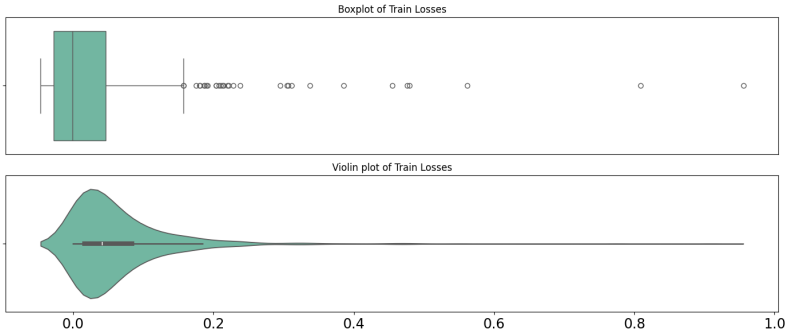


(d) Discriminator Loss

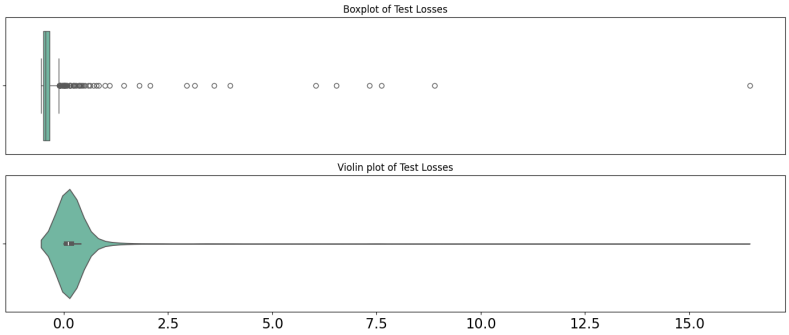


(e) Confusion Matrix

Figure C.25: Supplemental Visualizations for `wae-gan-punit15-1` - PUnit 15

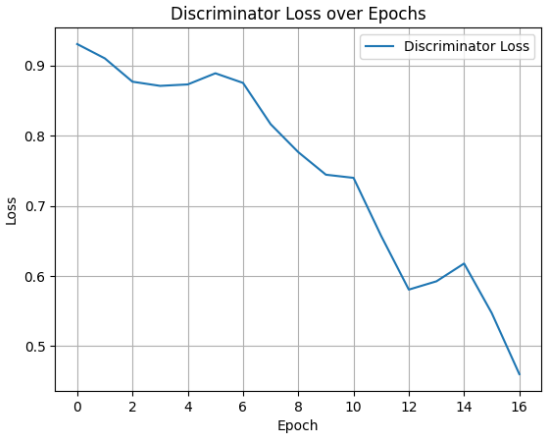


(a) Reconstruction Train Errors Distribution

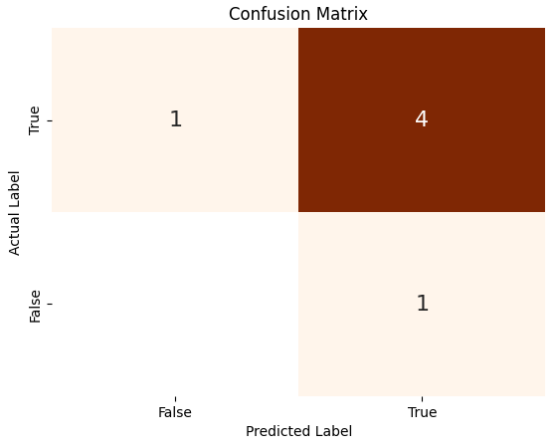


(b) Reconstruction Test Errors Distribution

(c) Reconstruction Loss

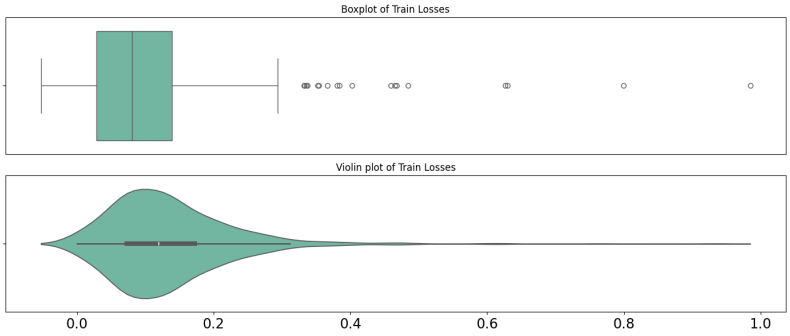


(d) Discriminator Loss

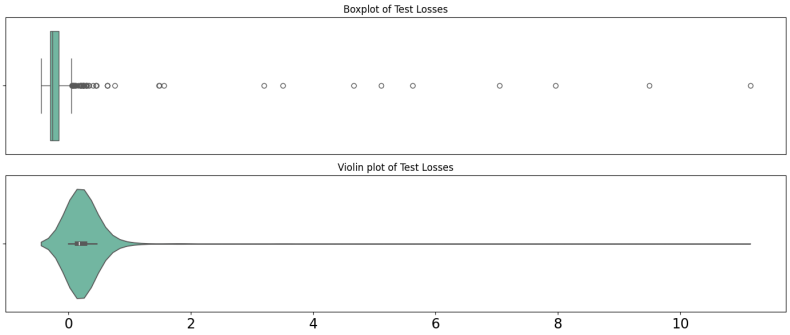


(e) Confusion Matrix

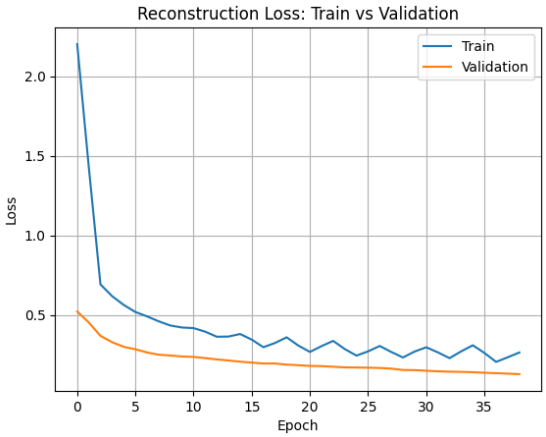
Figure C.26: Supplemental Visualizations for `wae-gan-punit15-2` - PUnit 15



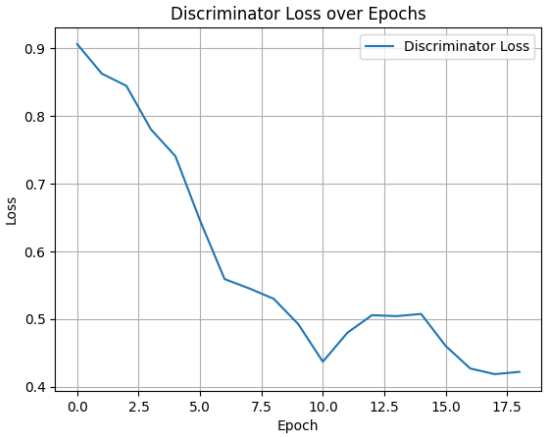
(a) Reconstruction Train Errors Distribution



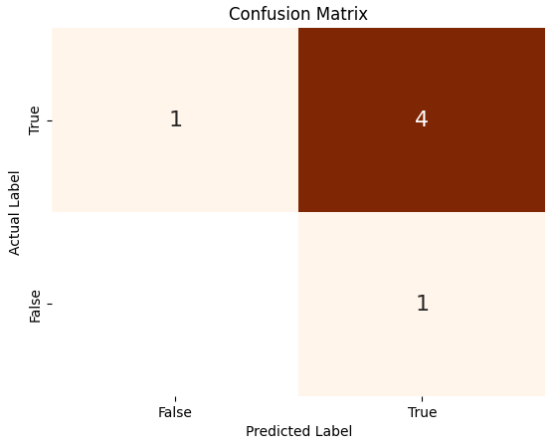
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss

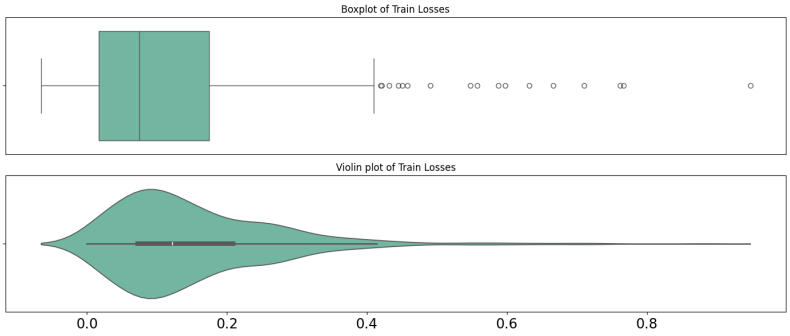


(d) Discriminator Loss

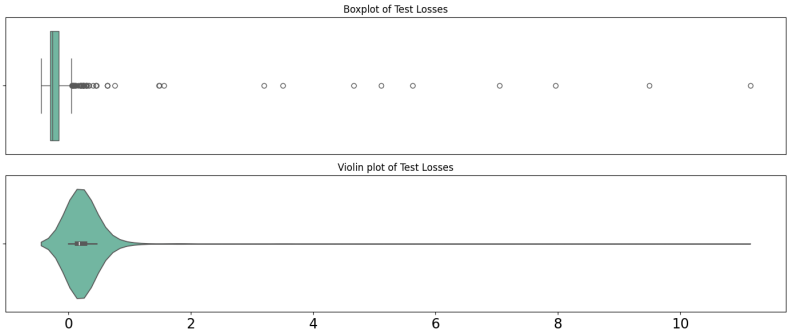


(e) Confusion Matrix

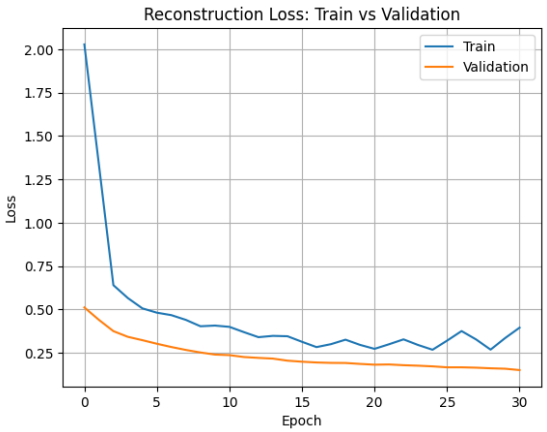
Figure C.27: Supplemental Visualizations for `wae-gan-punit15-3` - PUnit 15



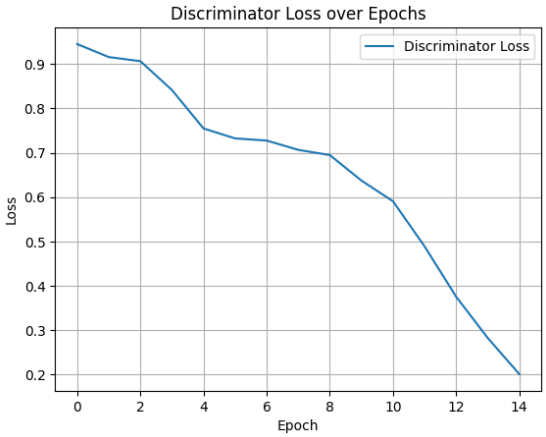
(a) Reconstruction Train Errors Distribution



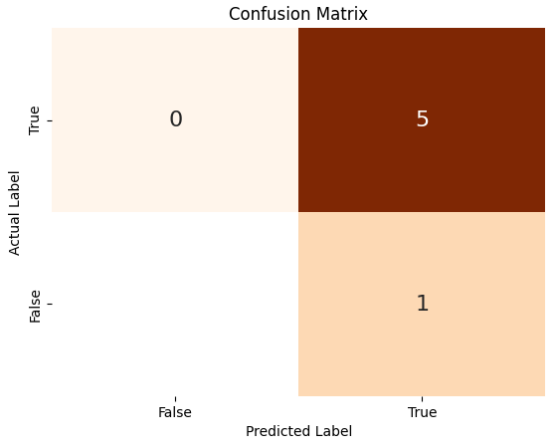
(b) Reconstruction Test Errors Distribution



(c) Reconstruction Loss



(d) Discriminator Loss

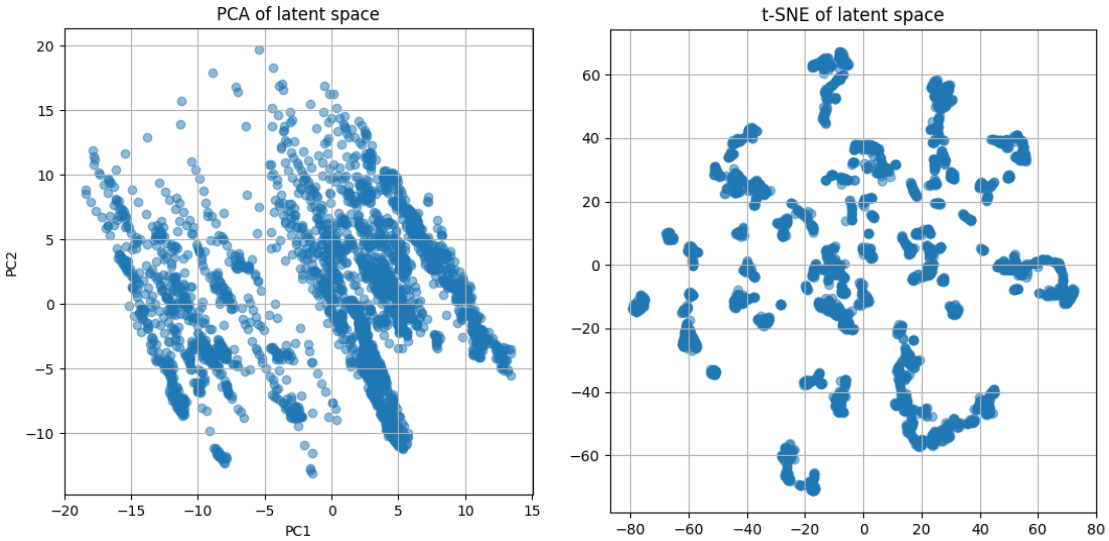


(e) Confusion Matrix

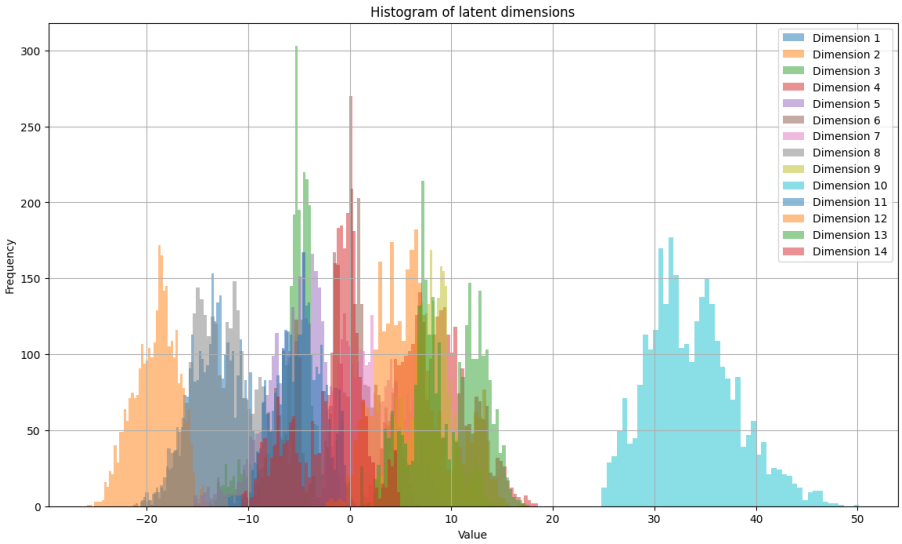
Figure C.28: Supplemental Visualizations for `wae-gan-punit15-4` - PUnit 15

C.3. WAE-GAN Latent Space for Optimal Models

The next Figures presents the latent space visualizations for the optimal WAE-GAN model identified for each PUnit (3, 10, 12, and 15), demonstrating the enforcement of a Gaussian distribution and the separation of normal and anomalous data.

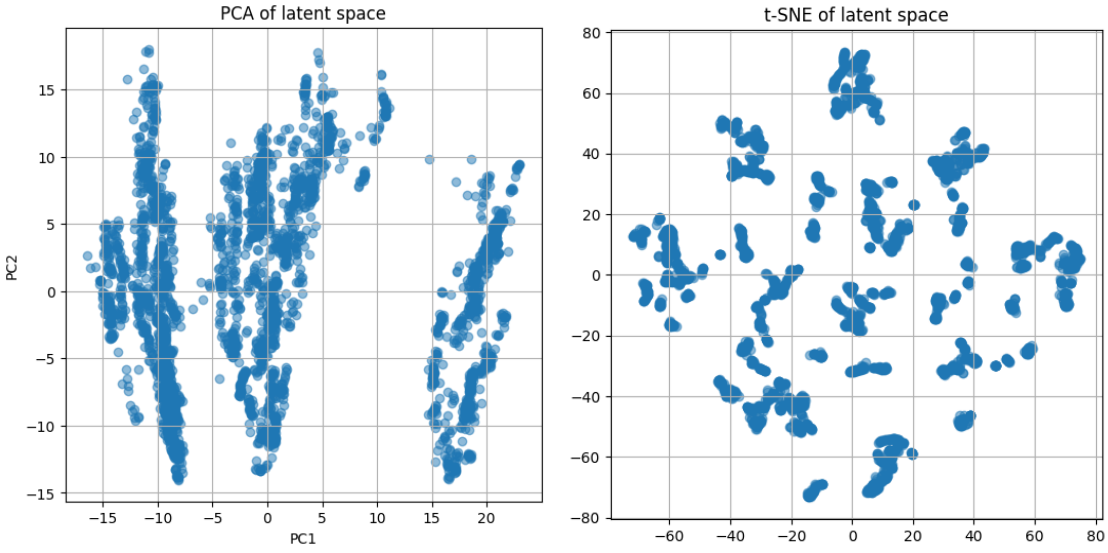


(a) PCA Latent Space (b) t-SNE Latent Space



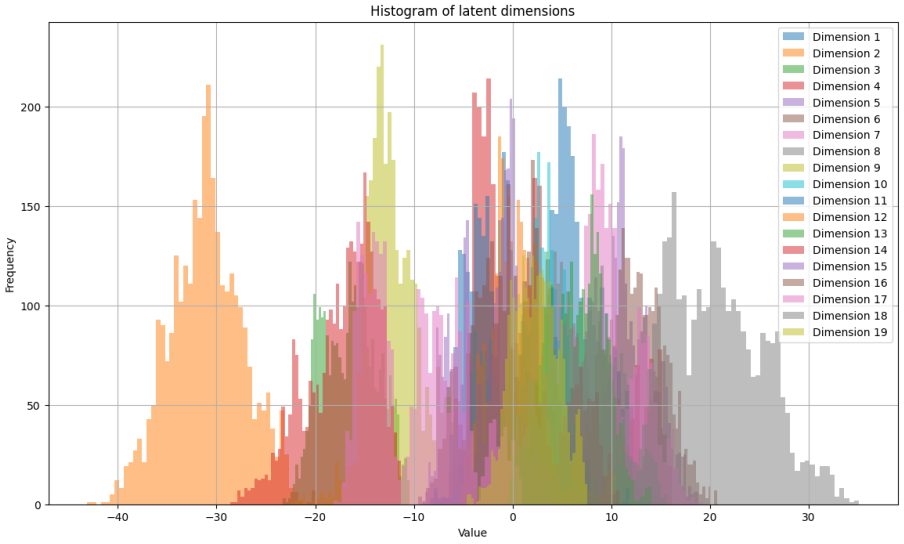
(c) Histograms of Latent Dimensions

Figure C.29: Latent Space Visualizations for Optimal WAE-GAN Model (`wae-gan-punit3-3`) - PUnit 3



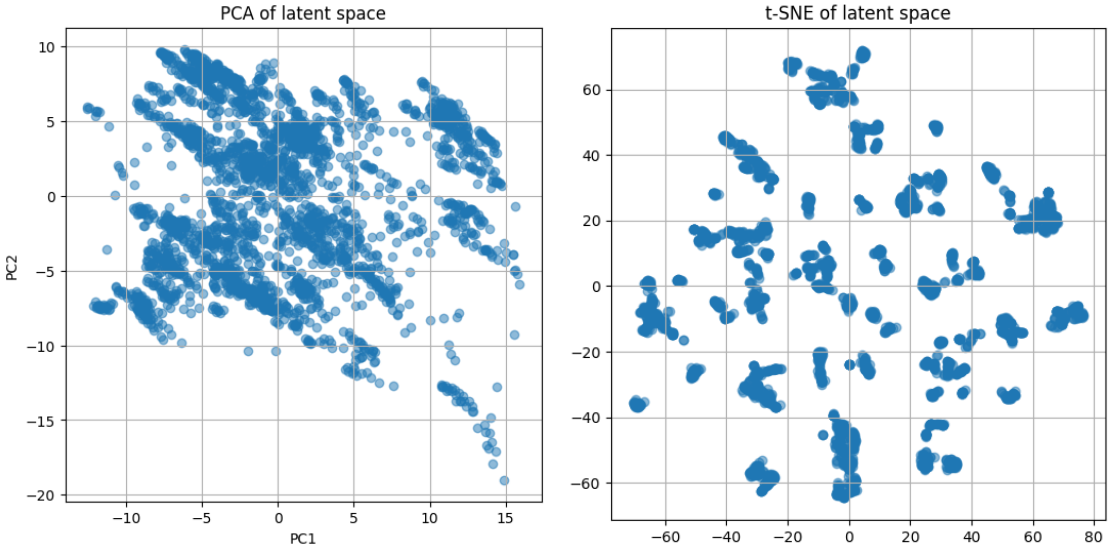
(a) PCA Latent Space

(b) t-SNE Latent Space



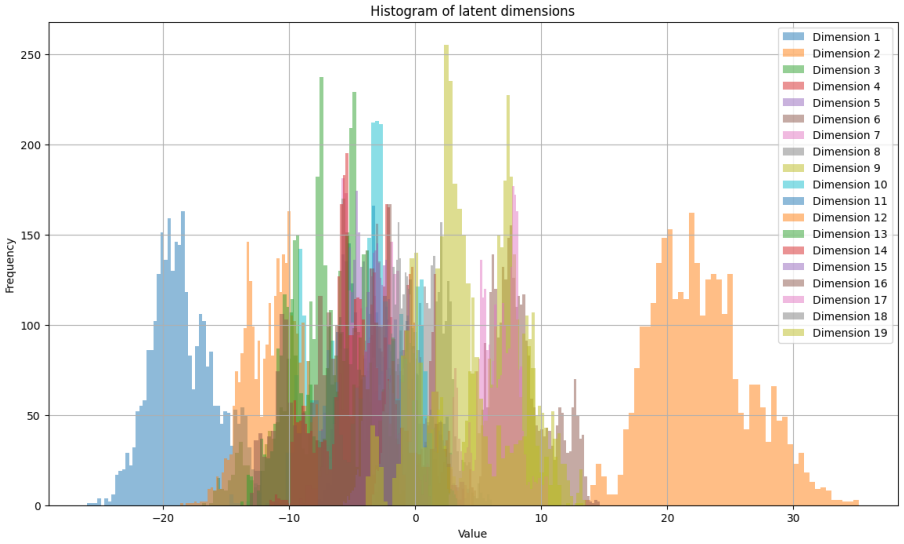
(c) Histograms of Latent Dimensions

Figure C.30: Latent Space Visualizations for Optimal WAE-GAN Model (`wae-gan-punit10-4`) - PUnit 10



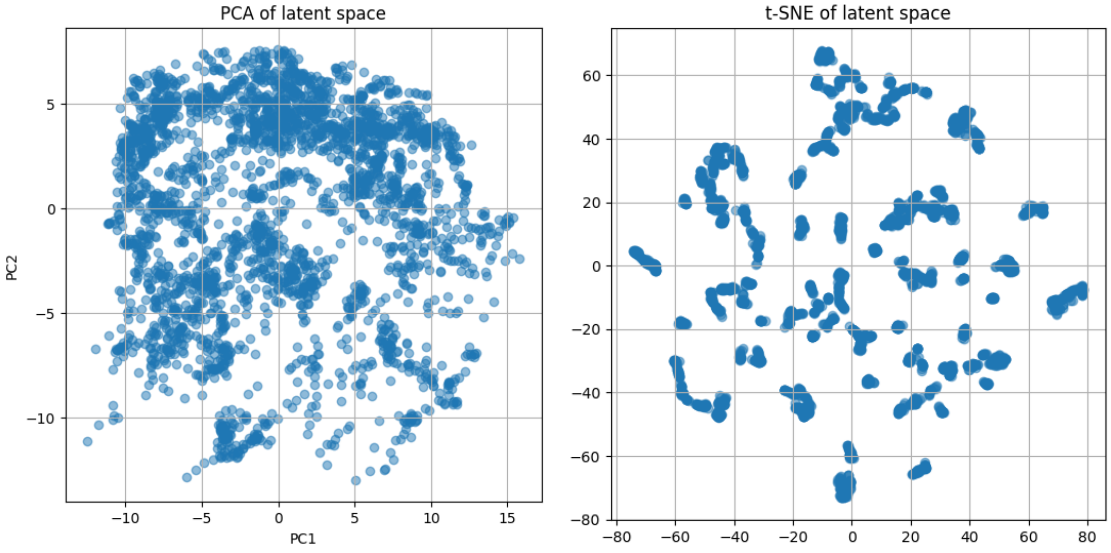
(a) PCA Latent Space

(b) t-SNE Latent Space



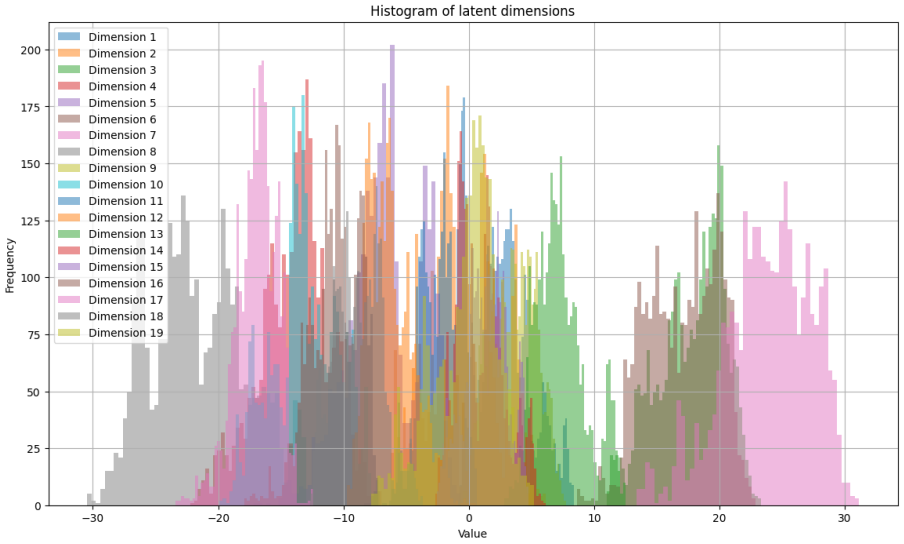
(c) Histograms of Latent Dimensions

Figure C.31: Latent Space Visualizations for Optimal WAE-GAN Model (`wae-gan-punit12-4`) - PUnit 12



(a) PCA Latent Space

(b) t-SNE Latent Space

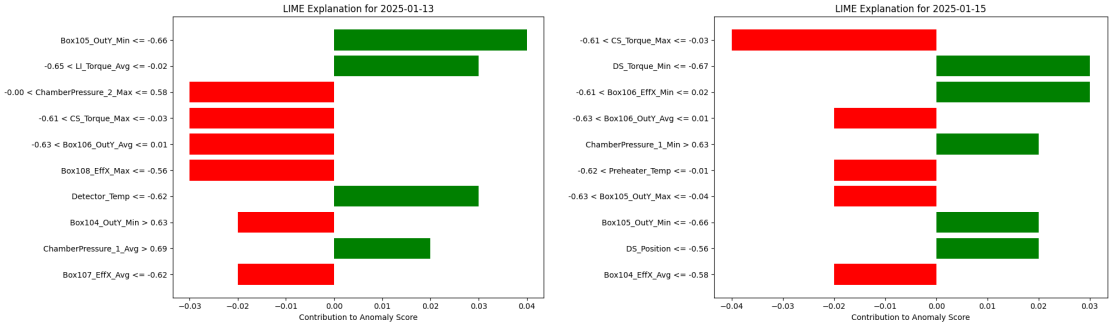


(c) Histograms of Latent Dimensions

Figure C.32: Latent Space Visualizations for Optimal WAE-GAN Model (`wae-gan-punit15-4`) - PUnit 15

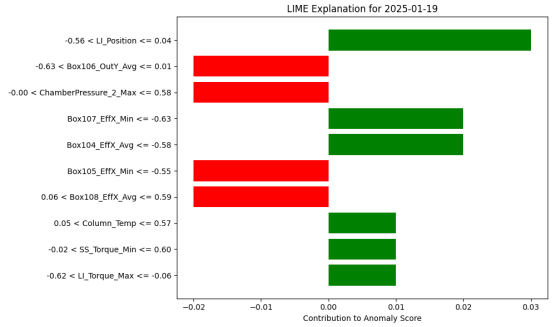
C.4. LIME Explanations

The next Figures present the LIME explanations for non-severe failures for each best model of each PUnit, showing the features that contributed positively and negatively to the failure detection, and this is very important to retrieve insights to the maintenance process.



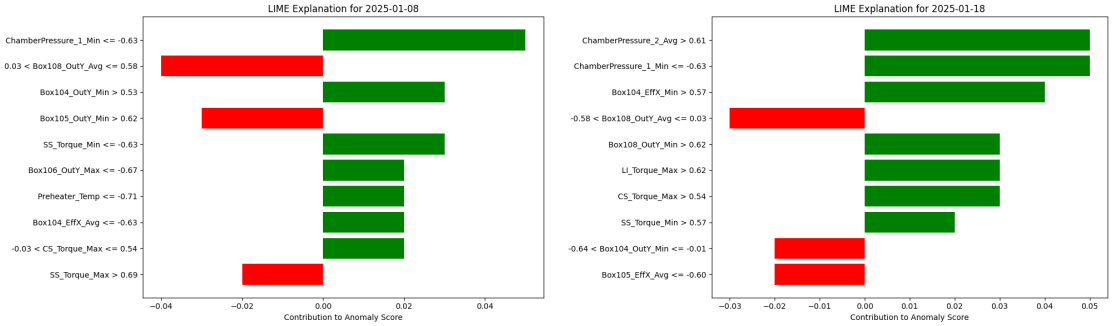
(a) Failure on 2025-01-13

(b) Failure on 2025-01-15



(c) Failure on 2025-01-19

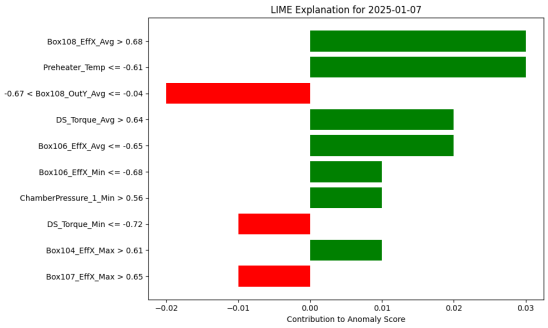
Figure C.33: LIME explanations for non-severe failures for model `wae-gan-punit3-3` - PUnit 3



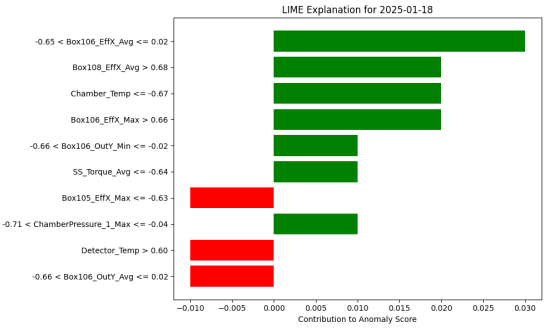
(a) Failure on 2025-01-08

(b) Failure on 2025-01-18

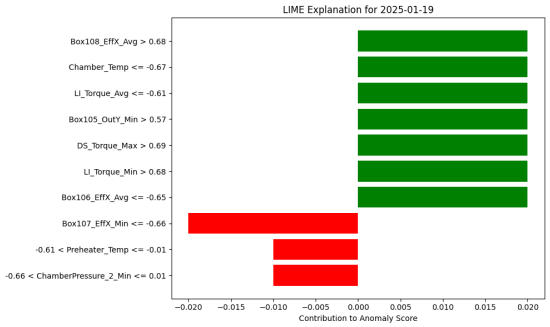
Figure C.34: LIME explanations for non-severe failures for model `wae-gan-punit10-4` - PUnit 10



(a) Failure on 2025-01-07

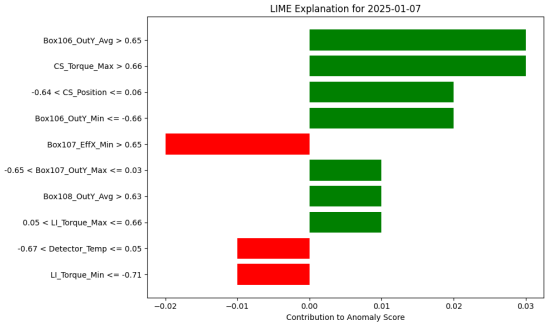


(b) Failure on 2025-01-18

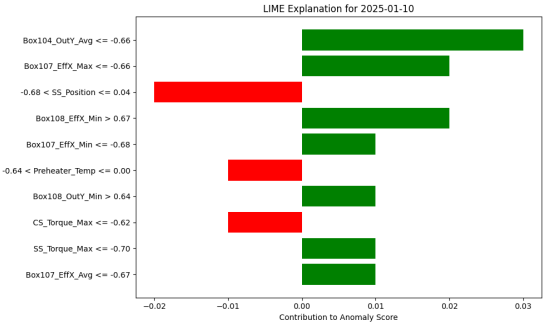


(c) Failure on 2025-01-19

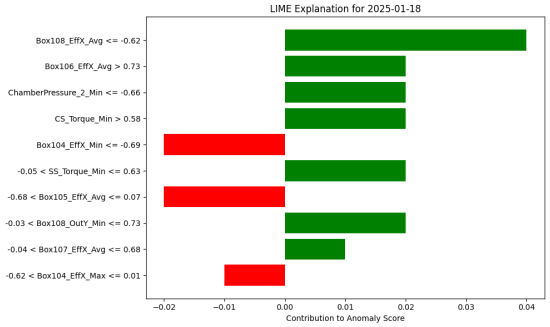
Figure C.35: LIME explanations for non-severe failures for model `wae-gan-punit12-4` - PUnit 12



(a) Failure on 2025-01-07



(b) Failure on 2025-01-10



(c) Failure on 2025-01-18

Figure C.36: LIME explanations for non-severe failures for model `wae-gan-punit15-4` - PUnit 15