CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Oscillation Detection on High-Resolution Time Series Data

Applying Anomaly Detection Techniques to Identify Oscillatory Segments in Vehicle Data

Master's thesis in Data Science and AI

Elvina Fahlgren

Albin Sundström

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Oscillation Detection on High-Resolution Time Series Data

Applying Anomaly Detection Techniques to Identify Oscillatory Segments in Vehicle Data

Elvina Fahlgren

Albin Sundström

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Oscillation Detection on High-Resolution Time Series Data
Applying Anomaly Detection Techniques to Identify Oscillatory Segments in Vehicle Data
Elvina Fahlgren
Albin Sundström

© Elvina Fahlgren, 2025.
© Albin Sundström, 2025.

Supervisor: Oana Geman, Department of Computer Science and Engineering
Advisor: Amanda Abed Ahad, Volvo Group
Examiner: Peter Damaschke, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Division of Data Science and AI
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Oscillation Detection on High-Resolution Time Series Data
Applying Anomaly Detection Techniques to Identify Oscillatory Segments in Vehicle Data
Elvina Fahlgren
Albin Sundström
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Field test vehicles at Volvo Group log between 500 and 1000 signals at resolutions up to 100 Hz, some of which originate from actuators. Oscillations in actuator signals can cause system instabilities and lead to components being worn out prematurely. This thesis investigates the suitability of unsupervised anomaly detection techniques for identifying such oscillations by framing them as a specific type of anomaly. Particularly, the focus is on evaluating the performance of a One-Class Support Vector Machine (OC-SVM) and a transformer-based model (TranAD). The available data is unlabeled and consists of high-frequency time series data collected from two main sources: field test vehicles and test cells. To complement this, a number of manual data recordings were provided by domain experts at Volvo Group, containing examples of oscillations. These recordings, combined with synthetically generated oscillations, were used to create a labeled test set. OC-SVM and TranAD were trained on both field test data and test cell data, with the best OC-SVM model being trained on field test data and the most effective TranAD model being trained on test cell data. Although both models are able to detect oscillations, they also capture other types of anomalies and sometimes misclassify normal data as anomalous. Overall, TranAD demonstrates the most promising result in detecting oscillatory behaviour. Since both OC-SVM and TranAD were able to detect oscillations, but also other types of anomalies, a valuable extension to this work would therefore be some sort of clustering as a postprocessing step. Despite some limitations, the models successfully identified oscillatory patterns that had not previously been detected at Volvo Group.

Keywords: anomaly detection, deep learning, machine learning, one-class support vector machine, transformer, unsupervised learning

Acknowledgements

We would like to acknowledge everyone who has helped and supported us throughout the course of this thesis project. We would like to thank our supervisor at Chalmers, Oana Geman, for her guidance and support.

We would also like to express our gratitude to everyone at Volvo Group who made our time there both productive and enjoyable. A special thanks goes to our advisor at Volvo Group, Amanda Abed Ahad, whose guidance was essential for this project.

Elvina Fahlgren, Albin Sundström, Gothenburg, June 2025

Contents

List of Figures xi

List of Tables xv

List of Acronyms xvii

Nomenclature xix

1 Introduction 1
- 1.1 Aim and Objectives . . . 2
- 1.2 Scope and Limitations . . . 2
- 1.3 Risks and Ethical Considerations . . . 2
- 1.4 Related Work . . . 3
- 1.5 Contributions . . . 5
- 1.6 Thesis Outline . . . 5

2 Theory 7
- 2.1 Actuators . . . 7
- 2.2 Time Series . . . 7
- 2.3 Anomaly Detection . . . 8
- 2.4 Data Preprocessing . . . 9
  - 2.4.1 Data Normalization . . . 9
  - 2.4.2 Sliding Window . . . 9
- 2.5 Machine Learning . . . 10
  - 2.5.1 Hyperparameter Tuning . . . 11
  - 2.5.2 One-Class Support Vector Machine . . . 11
- 2.6 Deep Learning . . . 13
  - 2.6.1 Artificial Neural Network . . . 13
  - 2.6.2 Transformer Architecture . . . 19
  - 2.6.3 TranAD . . . 22
- 2.7 Performance Metrics . . . 25
  - 2.7.1 Accuracy . . . 26
  - 2.7.2 Precision . . . 26
  - 2.7.3 Recall . . . 27
  - 2.7.4 F1 Score . . . 27
  - 2.7.5 ROC . . . 27
  - 2.7.6 AUC . . . 28

2.7.7 Range-Based Metrics . . . 29
2.7.8 IoU . . . 31

3 Data Description **33**
3.1 Field Test Data . . . 33
3.2 Test Cell Data . . . 33
3.3 Manual Data Recordings . . . 34
3.4 Manually Discovered Oscillations . . . 34

4 Methods **35**
4.1 Data Preprocessing . . . 35
4.1.1 Downsampling . . . 35
4.1.2 Signal Selection . . . 35
4.1.3 Normalization . . . 36
4.2 Data Sets . . . 36
4.2.1 Training Sets . . . 36
4.2.2 Synthetic Data Generation . . . 37
4.2.3 Test Sets . . . 44
4.3 Model Implementations . . . 45
4.3.1 One-Class Support Vector Machine . . . 46
4.3.2 TranAD . . . 48
4.4 Model Evaluation . . . 50
4.4.1 Window-Based . . . 51
4.4.2 Point-Based and Range-Based . . . 51
4.4.3 Segmentation and IoU Matching . . . 52

5 Results **53**
5.1 One-Class Support Vector Machine . . . 53
5.2 TranAD . . . 58

6 Discussion and Conclusion **67**
6.1 One-Class Support Vector Machine . . . 67
6.2 TranAD . . . 68
6.3 Limitations and Considerations . . . 70
6.4 Evaluation . . . 71
6.5 Future Work . . . 72
6.6 Conclusion . . . 73

Bibliography **75**

A Appendix: Training and Validation Loss **I**

B Appendix: Performance Results TranAD **V**

List of Figures

2.1 Example of the sliding window process applied to time series data, with window size $N = 4$ and a stride of 1. 10
2.2 Example of an artificial neural network architecture consisting of an input layer, three hidden layers and an output layer. Each circle represents a neuron. 14
2.3 Structure of a neuron. The inputs $x_1, x_2, x_3, \ldots, x_n$ are multiplied by their corresponding weights $w_1, w_2, w_3, \ldots, w_n$, and combined with a bias term b. The summation $\sum$ represents this weighted sum plus the bias, which is then passed through an activation function φ to produce the output a of the neuron. 15
2.4 Four standard activation functions in neural networks: sigmoid, softmax, tanh, and ReLU. 16
2.5 Forward propagation step for neuron k: inputs a_i, a_j, a_h are weighted by $w_{i,k}$, $w_{j,k}$, $w_{h,k}$ and summed to produce z_k, which is then passed through the activation function φ to receive the output a_k. This figure is adapted from Figure 5.2 in [49]. 17
2.6 Backward propagation for hidden neuron k: the error signal $\frac{\partial \mathcal{L}}{\partial a_k}$ is backpropagated by first computing the weighted sum of gradients from subsequent neurons, then multiplied by the derivative of the activation function $\frac{\partial a_k}{\partial z_k}$ to obtain δ_k. This figure is adapted from Figure 5.3 in [49]. 18
2.7 Transformer architecture, consisting of an encoder (left) and a decoder (right), each stacked N times. [54], CC-BY-SA 19
2.8 Attention mechanisms in transformers. Q, K, and V denote the query, key, and value vectors, respectively. 20
2.9 Overview of the TranAD architecture. $\vec{0}$ denotes the zero vector, C denotes context and W denotes the input window. PE refers to positional encoding. [23], CC-BY-NC-ND 22
2.10 Illustration of a confusion matrix for a binary classification task. The matrix includes true positives, false positives, true negatives and false negatives. 26
2.11 Example of a ROC curve showing the true positive rate versus the false positive rate [65], CC-BY-SA 28
2.12 Range-based evaluation example. 31

3.1 Example of an oscillation found in the test cell data. The blue line shows the oscillating actuator, while the green line shows the engine torque and the orange line shows the engine speed. 34

4.1 Example of a manual data recording of an oscillation in an actuator in 80 Hz. The data was downsampled to 10 Hz, preserving the overall pattern despite some minor loss of detail. 36
4.2 Two synthetic time series samples of constant sine and cosine waves, of lengths 50. 38
4.3 Two time series samples of sine waves with linear increase and decrease amplitude variation functions applied. Both time series are of length 50. 39
4.4 Two time series samples of sine waves with exponential increase and decrease amplitude variation functions applied. Both time series are of length 50. 40
4.5 Two time series samples of sine waves with sinusoidal and cosine amplitude variation functions applied. Both time series are of length 50. 40
4.6 Two time series samples of sine waves with random and logistic amplitude variation functions applied. Both time series are of length 50. 41
4.7 Two time series samples of sine waves with quadratic and square root amplitude variation functions applied. Both time series are of length 50. 42
4.8 Two time series samples of sine and cosine waves with applied amplitude variation and trend functions. Both time series are of length 50. 43
4.9 Two samples of segments of the original time series and corresponding modified time series with inserted oscillations, used as test set. The blue dashed lines correspond to the original time series, while the orange lines correspond to the modified time series. 45

5.1 Two examples of true positive detections by OC-SVM based on pointwise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold. 55
5.2 Two examples of false negative cases from OC-SVM based on pointwise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold. 56

5.3 Three examples of false positive cases from OC-SVM based on pointwise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold. 57

5.4 Comparison of oscillation predictions and anomaly scores from TranAD models trained on field test data and test cell data. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold. 58

5.5 Density distributions of window-level anomaly scores for Vehicle 1 and Vehicle 4 in the datasets containing 1% manually discovered oscillations. The vertical dashed line indicates the threshold obtained using POT. 59

5.6 Three examples of true positive detections by TranAD based on pointwise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold. 61

5.7 Two examples of false negative cases from TranAD based on pointwise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold. 62

5.8 Three examples of false positive cases from TranAD based on pointwise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold. 63

5.9 Predictions from TranAD using POT with a stricter threshold, $p = 0.90$, $q = 10^{-5}$. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line represents the threshold. . 64

5.10 Comparison of oscillation predictions and anomaly scores from multivariate and univariate TranAD. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line represents the threshold. 65

5.11 Comparison of oscillation predictions and anomaly scores from multivariate and univariate TranAD. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line represents the threshold. 66

A.1 Training and validation loss for TranAD trained on univariate field test data. The blue line shows training loss, while the orange line shows validation loss. Peaks in validation loss correspond to transitions between vehicles in the training sequence. I
A.2 Training and validation loss for TranAD trained on univariate test cell data. The blue line shows training loss, while the orange line shows validation loss. Small number of epochs due to early stopping. II
A.3 Training and validation loss for TranAD trained on multivariate test cell data. The blue line shows training loss, while the orange line shows validation loss. III

List of Tables

3.1 Overview of the vehicle signals used in this thesis, including their corresponding units and data types. 33

4.1 Overview of the training sets, including their corresponding data sources, sizes and the models they were used for. 37
4.2 Configuration table for synthetic time series generation. 43
4.3 Number of data points per vehicle time series. 44
4.4 Hyperparameter search space used in Optuna tuning. 48
4.5 Final hyperparameter configuration for OC-SVM with the RBF kernel. 48
4.6 Final hyperparameter configuration for OC-SVM with the Polynomial kernel. 48
4.7 Training configuration for TranAD 50
4.8 Hyperparameter search space used in Optuna optimization 50

5.1 Window-level performance of OC-SVM over the two test set variants across the four vehicle time series. Best results are shown in bold. . . 53
5.2 Performance of OC-SVM over the two test set variants across the four vehicle time series. P_X and R_X denote range-based precision and recall, computed from point-level labels. TP, FN, and FP are segment-level counts. Best range-based results are in bold. 54
5.3 Window-level performance of TranAD over the two test set variants across the four vehicle time series. Best results are shown in bold. . . 59
5.4 Performance of TranAD on the two test set variants across the four vehicle time series. P_X and R_X denote range-based precision and recall, computed from point-level labels. TP, FN, and FP are segment-level counts. The best range-based results are shown in bold. 60

B.1 Window-level performance of TranAD, with threshold from POT with parameters $p = 0.90$ and $q = 10^{-5}$, over the two test set variants across the four vehicle time series. Best results are shown in bold. . . V
B.2 Performance of TranAD, with threshold from POT with parameters $p = 0.90$ and $q = 10^{-5}$, over the two test set variants across the four vehicle time series. P_X and R_X denote range-based precision and recall, computed from point-level labels. TP, FN, and FP are segment-level counts. The best range-based results are shown in bold. V

List of Acronyms

Below is a list of acronyms that are used throughout this thesis, listed in alphabetical order:

AD	Anomaly Detection
AUC	Area Under the Curve
FN	False Negative
FNN	Feed-Forward Neural Network
FP	False Positive
GPD	Generalized Pareto Distribution
IAE	Integrated Absolute Error
IoU	Intersection over Union
ML	Machine Learning
OC-SVM	One-Class Support Vector Machine
POT	Peaks-Over-Threshold
RBF	Radial Basis Function
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SV	Support Vector
SVM	Support Vector Machine
TN	True Negative
TP	True Positive

Nomenclature

Below is a nomenclature of functions, indices, sets, parameters, and variables that are used throughout this thesis.

Functions

$\mathcal{L}$	Loss function
φ	Activation function
P_X	Range-based precision
R_X	Range-based recall

Indices

t	Index for time step in a time series

Sets

$\mathcal{T}$	Set of time indices
$\mathcal{R}$	Set of labeled anomaly ranges
$\mathcal{P}$	Set of predicted anomaly ranges

Parameters

α	Weight assigned to the existence reward component in range-based recall
β	Weight assigned to the overlap reward component in range-based recall
γ	Kernel hyperparameter in OC-SVM

ν	Fraction of outliers parameter in OC-SVM
η	Learning rate
p	Percentile level for initial POT threshold
r	Initial POT threshold for tail modeling
q	POT risk level for anomaly decision
T	Time series length

Variables

s_{x_t}	Point-level anomaly score at time step t
s_{W_t}	Window-level anomaly score at time step t
x_t	Data point at time step t
W_t	Window ending at time step t
X	Time series sequence

1

Introduction

Heavy-duty vehicles contribute to more than 25% of CO2 emissions from road transport in the European Union [1]. In response, the European Commission introduced the European Green Deal in 2019, aiming to reduce greenhouse gas emissions across the region [2]. Volvo Group has committed to reducing CO2 emissions in operations by 50% by 2030 and achieving climate neutrality in operations by 2040, from cradle to gate. A part in achieving this will be ensuring that vehicles in production operate as intended. This presents a challenge, as some issues are nearly impossible to detect manually. To address this, the use of data has become a more common approach for identifying potential problems.

The use of high-resolution time series data logging in vehicles at Volvo Group provides an opportunity to detect patterns and anomalies that may indicate system failures or abnormal behaviors. Field test vehicles capture data at resolutions up to 100 Hz, logging between 500 and 1000 signals from various control units. These control units have a critical role in managing and controlling different components in the truck, such as the engine and the gearbox. Some of the logged signals originate from actuators, which convert signals into physical movement, for example by adjusting a valve. Oscillations in actuator signals can cause system instabilities and, over time, lead to components being worn out prematurely. The logs are collected to validate and verify new features and functionalities, offering detailed insights that could help identify unexpected patterns. Similar data is logged in so called test cells, controlled environments where engines are tested during the development stages. Both field test data and test cell data are highly complex, as they involve numerous parts interacting in ways that affect each other's behavior. This complexity makes detecting anomalies, such as oscillatory patterns, especially challenging. According to domain experts at Volvo Group, anomalous behavior is currently detected as engineers manually inspect data and coincidentally find unexpected patterns, such as oscillations. While this approach has successfully identified certain anomalies, it is inefficient, and there is a risk that oscillations could go unnoticed.

Anomaly detection offers a more systematic and scalable alternative for identifying such patterns in time series data. While anomaly detection methods in time series data have received significant attention, the specific problem of detecting oscillatory anomalies with such methods is less commonly addressed. This thesis aims to address that gap by evaluating two unsupervised models on real-world actuator

data.

1.1 Aim and Objectives

The goal of this thesis is to develop and evaluate anomaly detection techniques capable of identifying patterns and anomalies, in the form of oscillations, in high-resolution time series data logged by field test vehicles. The main objectives of the thesis are:

- Develop two unsupervised anomaly detection models, one traditional machine learning model (One-Class SVM) and one more advanced transformer-based model (TranAD).
- Evaluate the performance of the models and assess the suitability of anomaly detection techniques for this use case.

The evaluation should answer the following questions:

- How well does the models perform, given the presence of unlabeled, naturally occurring oscillations in the data?
- To what extent are unsupervised anomaly detection methods suitable for detecting oscillations in real-world actuator data?

1.2 Scope and Limitations

The aim of the thesis is not to develop an anomaly detection model of optimal performance, but rather to test the feasibility of anomaly detection models on the given dataset. The scope of the thesis is focused on oscillations, not anomalies in general. However, given that anomaly detection techniques are used, the same principles should be applicable to different types of anomalies. Moreover, the number of models to be investigated has been limited to two, and the study is focused on data from one actuator. Furthermore, it was assumed that no considerable computational limitations were linked to the objective of this thesis, and thus, computational cost will not be investigated.

The available dataset is unlabeled, which makes evaluation more difficult. There exist some examples of oscillations that will be used for the evaluation of the models. However, since the examples are limited, the results will likely not be an entirely accurate representation of the true performance of the models.

1.3 Risks and Ethical Considerations

The dataset used for this thesis contains some personal data. The data is logged by test vehicles, where customers of Volvo Group have agreed to share data for research and development purposes. During the preprocessing, signals relevant for this thesis

will be selected, and any signals that are considered personal data should be filtered out after that stage. In addition, handling of personal data should always be done with caution, to respect the privacy of the customers and to comply with GDPR and other data protection laws.

Our contribution towards Volvo Group's climate goals has been to improve the capability to detect oscillations in a more efficient way, saving valuable time and resources, and most importantly to help ensure that the final products are used as intended. In turn, this will then improve the performance of the final product, leading to lower emissions and fuel consumption.

ChatGPT and similar AI tools have been used for text refinements and code debugging. All use has been in accordance with Chalmers regulations for the use of AI tools in thesis work [3].

1.4 Related Work

Oscillation detection is a common and critical challenge in systems with control loops. A control loop is a feedback-based system that continuously adjusts its inputs in order to maintain a desired output [4]. Such systems are particularly common in process industries where oscillations may negatively impact product quality, increase energy consumption and waste raw materials [5]. Similar challenges arise in actuators, which function as control loops, where oscillations can indicate mechanical issues or external disturbances. While much of the existing research focuses on oscillation detection in control loops in process industries, the methods are also relevant in the context of actuators. Traditional methods include IAE-based methods, which compute the Integrated Absolute Error between zero-crossings and detect oscillation based on a threshold [6], [7], as well as frequency-domain approaches such as Fourier analysis [8] and wavelet transforms [9]. Additionally, decomposition-based methods, such as Empirical Mode Decomposition (EMD), have been applied to isolate oscillatory components from noisy signals [5].

Despite the simplicity and effectiveness of traditional methods, they often rely on predefined criteria and require prior knowledge of signal characteristics such as frequency ranges and noise levels. These dependencies limit their applicability in complex and noisy environments, which are common in industrial data where frequency and amplitudes of oscillations may be intermittent. To address these limitations, Dambros *et al.* [10] propose a deep learning-based approach to automatically detect oscillations in process industries data. Their model is a deep feed-forward network, trained on artificially generated data designed to represent both oscillatory and non-oscillatory signals, including a wide range of characteristics such as disturbances, oscillation waveforms, and frequencies. Being evaluated on both artificial and measured data, their model demonstrates its robustness to varying process conditions such as noise, disturbances, and intermittent oscillations. This highlights the potential of deep neural networks to provide effective solutions for oscillation detection.

In addition to deep learning-based approaches, recent research has explored alternative strategies for oscillation detection. Memarian *et al.* [11] propose a shape-based pattern recognition method to identify oscillatory control loops in process industries. Their approach leverages the geometric properties of controller output and process variable signals, identifying oscillations by detecting a distinct triangle-like shape in the data. Two different methods were used: an unsupervised method based on nonlinear algebraic function fitting and a deep Convolution Neural Network (CNN) trained to recognize oscillatory patterns. Their results demonstrated the effectiveness of both methods.

While a range of oscillation detection methods exists, they are often tailored to specific process characteristics or rely on labeled training data, both of which limit their flexibility in complex environments. The application of unsupervised machine learning methods, specifically for oscillation detection, remains limited.

An alternative perspective is to frame oscillations as *anomalies*, deviations from expected system behavior [12]. This framing allows oscillation detection to be approached within the field of anomaly detection, focusing on identifying unusual patterns without prior knowledge of their form. Anomaly detection in time series data is a well-studied field and has been used in a wide range of applications, including intrusion detection in cybersecurity, faults in manufacturing equipment, and leaks in gas-chemical processes [13]. The main objective of anomaly detection is to identify observations or segments that deviate from expected patterns, often without knowing the precise nature of these anomalies [12].

Machine learning-based approaches for the detection of anomalies in time series data have been a popular choice across various fields [13], [14]. One of the most common machine learning methods is One-Class Support Vector Machine (OC-SVM), originally proposed by Schölkopf *et al.* [15], which works by defining a boundary around normal data points and classifying observations outside this boundary as anomalies. It has been used as a baseline approach in several time series anomaly detection studies, including the detection of unusual patterns in airplane communication systems, failure detection in network systems, and identification of irregular heartbeats in medical data [16], [17], [18].

More recently, the use of deep learning models, such as Recurrent Neural Networks (RNNs), Autoencoders (AEs), Long Short-Term Memory (LSTM) and transformer networks have emerged to address the challenges of capturing temporal dependencies and learning richer representations for anomalies in time series data [19], [20], [21], [22]. These deep learning methods have become particularly popular due to their ability to model complex data dynamics without relying on assumptions about underlying patterns [13]. One deep learning model that has shown promising results across multiple datasets is the transformer-based network TranAD, specifically designed for anomaly detection in time series data [23]. It also proves effective in network monitoring, detecting issues like traffic overload attacks with high accuracy and fast processing [17], and other domains, such as avionics [16]. The model leverages the transformer architecture and incorporates adversarial training to improve

robustness [23]. TranAD is trained using a reconstruction-based approach, meaning it learns to reconstruct normal sequences and identifies anomalies by assigning higher reconstruction errors to anomalous segments.

1.5 Contributions

Despite advances in anomaly detection, the specific challenge of identifying oscillatory anomalies using machine learning remains largely underexplored, particularly in the context of unsupervised learning. This thesis addresses this gap by framing oscillations as a specific type of anomaly in time series data and by evaluating the effectiveness of two unsupervised approaches: OC-SVM and TranAD. The models are applied to measured actuator data, offering new insights into the suitability of unsupervised anomaly detection models for oscillation detection in complex environments.

1.6 Thesis Outline

The paper is structured as follows:

- Chapter 2 introduces key concepts and provides a theoretical background necessary for understanding the methods used in this work.
- Chapter 3 presents the available data, including their characteristics and relevant structural properties.
- Chapter 4 outlines the methodology, including data preprocessing steps, dataset preparation, model implementations and evaluation strategies.
- Chapter 5 presents the experimental results and outlines the performance of the models.
- Chapter 6 discusses the results, suggests possible directions for future work and concludes the thesis.

2

Theory

This chapter outlines the theoretical background behind the concepts, methods and models applied throughout this thesis. It begins with a brief description of actuators in Section 2.1. Then, in Section 2.2, the principles of time series are introduced. Section 2.3 focuses on anomaly detection in the context of time series. In Section 2.4, relevant data preprocessing techniques are explained. Section 2.5 introduces the concept of machine learning and explains learning approaches, hyperparameter tuning, and OC-SVM. Section 2.6 presents the fundamentals of deep learning, including how artificial neural networks work, and introduces transformers and TranAD. Finally, Section 2.7 explains performance metrics relevant for this work.

2.1 Actuators

An actuator is a component that converts signals into physical movement or action [24]. There are various types of actuators, each serving different functions across a wide range of applications. For instance, an actuator connected to the air intake valve regulates the air intake into the engine. The position of an actuator can be influenced by a multitude of variables, such as engine speed and torque. This complexity makes it difficult to detect anomalies in actuator behavior, especially when multiple factors interact with each other.

2.2 Time Series

A time series is defined as a sequence of data points $\{x_t\}$ observed over an ordered time interval, indexed by time $t \in \mathcal{T}$, for example $\mathcal{T} = \{1, 2, \ldots, T\}$, where T denotes the total number of time steps [25]. Each data point x_t represents a measurement taken at a specific point in time. Time series can be categorized by dimensionality, which refers to the number of variables recorded at each time step [13]. According to this criterion, time series are commonly divided into two types: univariate and multivariate.

A univariate time series consists of an ordered set of real-valued observations $\{x_t\}_{t \in \mathcal{T}}$, where each $x_t \in \mathbb{R}$ is a single measurement at time t. This type captures the changes of a single variable over time and focuses primarily on modeling temporal dependencies [13]. In contrast, a multivariate time series extends this concept by

observing multiple variables at each time point. Each observation is a vector $x_t \in \mathbb{R}^d$, forming a sequence $X = \{x_t\}_{t \in \mathcal{T}}$, where d is the number of dimensions. In practice, multivariate time series can be interpreted as a collection of interrelated univariate series, representing the behavior of a system through combinations of several variables at each time step [13].

2.3 Anomaly Detection

Anomaly detection (AD) refers to the task of identifying data points or segments that deviate from established patterns of normal behavior [12]. In the context of time series data, AD refers to identifying irregular behavior over time, where anomalies are defined as data point(s) at specific time step(s) that deviate from previously observed normal behavior [13]. AD in time series data has been applied in a variety of fields, including detecting faults in manufacturing, leaks in chemical processes, cyber intrusions, and other unexpected system disturbances [13].

As categorized by [13], anomalies in time series can be divided into three main types based on their behavior over time and the context in which they occur. The first type is referred to as a point anomaly, representing a single data point or short sequence that sharply deviates from the expected pattern, often due to sensor faults or sudden system errors. The second type is a contextual anomaly, where a data point may fall within a normal range globally but appears anomalous given its surrounding context. The third type is a collective anomaly, where a time series segment shows anomalous behavior, but the individual points within that segment are not necessarily point anomalies. In addition to these categories, other perspectives may define more specific anomaly types depending on what is defined as "normal" behavior in a given application domain [13]. In the context of this work, the focus is on finding collective anomalies, where irregular patterns span across multiple time steps.

Many anomaly detection models output a continuous anomaly score rather than a binary label [14]. A higher score often means the point or segment is considered more anomalous. These scores can then be thresholded to identify anomalies.

Traditionally, techniques that have been applied in time series AD rely on different statistical and mathematical models [13]. These include time frequency domain analysis, such as Fourier analysis; statistical techniques, such as measuring mean, variance and median; and distance-based models that compute similarities between sequences. Moreover, clustering algorithms such as k-means, DBSCAN, One-Class Support Vector Machines (OC-SVM) and Gaussian Mixture Models (GMM) are also popular techniques in time series AD [13]. In addition, more recent approaches are deep learning methods. These have been introduced to model more complex temporal dependencies and high-dimensional data [13]. Architectures such as Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), Autoencoders (AEs), and transformers are commonly used. These models typically learn representations of normal behavior and assign anomaly scores based on deviations from expected patterns, making them effective for detecting complex and multivariate

anomalies in time series data.

2.4 Data Preprocessing

Data preprocessing refers to the transformation of raw data into a structured and standardized format suitable for machine learning and statistical analysis [26]. It aims to preserve valuable information, solve data issues that could prevent analysis of the data, and make the data more useful for subsequent processing. This section describes two such techniques: normalization and sliding window segmentation.

2.4.1 Data Normalization

Normalization is a commonly used preprocessing step, where data is rescaled to a common scale [27]. This ensures that all features are appropriately weighted, preventing any single feature from dominating due to its scale. It also improves convergence and numerical stability.

Min-Max Normalization

Min-max normalization rescales data to the interval [0,1]. This is done by subtracting the minimum value of the feature and dividing by the range, which is the difference between the maximum and minimum values. The result is that all values are mapped to the interval [0,1], while the distribution shape is preserved. It is mathematically defined as:

$$x' = \frac{x - \min(Z)}{\max(Z) - \min(Z)}, \tag{2.1}$$

where x is a single data point of a specific feature, Z is the set of all values of that specific feature and x' is the scaled value.

2.4.2 Sliding Window

The sliding window technique is a commonly used method when working with time series data [28]. It is particularly useful when it comes to capturing anomalous subsequences in time series, as it enables the detection of entire segments that deviate from expected patterns [29]. Since time series data consists of sequential observations, their values depend on their local context. The sliding window technique works by structuring the data into segments, which enables models to learn and recognize patterns over time [30]. Consider a time series:

$$X = \{x_1, x_2, \ldots, x_T\},$$

where $x_t \in \mathbb{R}^d$ represents an observation at time step t, T is the total number of time steps, and d is the dimension of the time series. The sliding window technique

partitions X into subsequences, or windows, of fixed length N by sliding forward by a fixed step size called stride [30]. Each window W_t is defined as:

$$W_t = \{x_{t-N+1}, x_{t-N+2}, \ldots, x_t\}$$

for $t = N, N+1, \ldots, T$, assuming a stride of one, which ensures maximal overlap. Each $W_t \in \mathbb{R}^{N \times d}$ captures a segment of the time series for use in model training or analysis. Figure 2.1 shows the process of the sliding window technique, visualizing how a window slides across a time series.

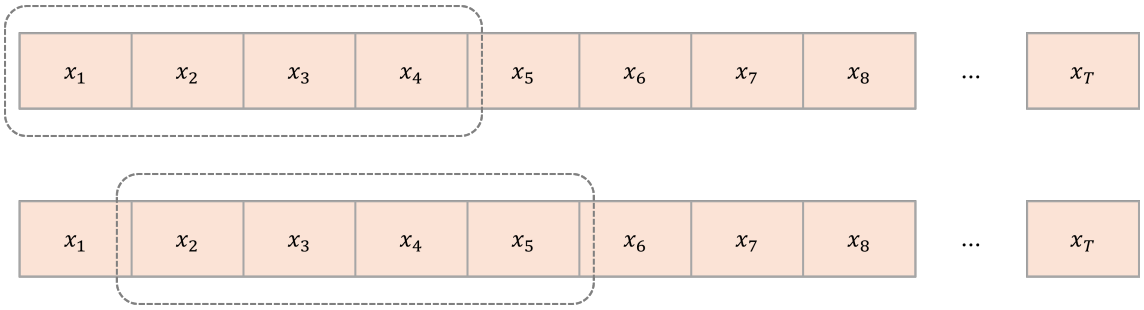


Figure 2.1: Example of the sliding window process applied to time series data, with window size $N = 4$ and a stride of 1.

When deciding the size of the window, several trade-offs must be considered. The window must be large enough to capture the relevant temporal dependencies in the data while remaining small enough to avoid unnecessary computational costs and smoothing of local variations [5]. This balance is particularly challenging in practice, as the optimal window size is dependent on the problem.

2.5 Machine Learning

Machine learning (ML) is a central area within artificial intelligence (AI) focused on developing systems that can learn from data and improve their performance over time [31]. Instead of relying on manual rules, ML models are designed to recognize patterns in data and make decisions based on past observations. Mitchell [32] defines machine learning as:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [32, p.2].

This definition highlights the key components of a learning problem: the data that the model learns from, the task, and the measure used to evaluate performance. In contrast, Kelleher [33] expresses this idea in mathematical terms, saying that the goal of machine learning is to approximate a function that maps inputs to outputs [33]. This function captures the patterns in the data and allows the model to make predictions or decisions.

Supervised learning is one of the most widely used approaches in machine learning, where models learn from labeled data by mapping inputs to known outputs [34].

However, in many real-world scenarios, labeled data is limited or expensive to obtain, while unlabeled data is often abundant and readily available. Unlike supervised learning, unsupervised learning does not rely on labeled data. Instead, models learn patterns and structures in the data without any labels. By uncovering hidden relationships, grouping similar data points, or reducing dimensionality, unsupervised learning is used for tasks such as clustering, anomaly detection and feature learning [34].

2.5.1 Hyperparameter Tuning

Most machine learning models have some configurable variables, called hyperparameters, which affect their performance [35]. Therefore, hyperparameter tuning is an essential part of machine learning to optimize performance. Common approaches to hyperparameter tuning include grid search and random search, however, both of these have major drawbacks [36]. Grid search exhaustively searches a user-defined finite set of hyperparameter values and evaluates the model using each combination in the defined set. Since the set of combinations grows exponentially, grid search suffers from the curse of dimensionality. Although grid search is easy to use, the mentioned drawbacks make it inefficient for large search spaces. Random search is an alternative approach to grid search, which randomly samples hyperparameter combinations, instead of doing an exhaustive search like grid search. When a few hyperparameters are significantly more important than others, random search is better than grid search. Since random search explores a smaller subset of hyperparameter combinations than grid search, it operates more efficiently, especially in high-dimensional search spaces. However, unlike grid search, random search is not guaranteed to find the optimal solution [37].

More advanced alternatives to the relatively simple grid search and random search include hyperparameter optimization frameworks, such as Optuna [38]. This framework offers greater flexibility, as it supports multiple optimization algorithms [39], [40]. Optuna enhances hyperparameter tuning by incorporating both advanced sampling algorithms and pruning strategies. Unlike grid search and random search, which evaluate all trials equally, Optuna incorporates pruning algorithms to dynamically adjust the search process and terminate unpromising trials at an early stage [38]. Additionally, Optuna provides access to more efficient sampling methods, such as Tree-structured Parzen Estimator (TPE) and Bayesian optimization, which focus on promising regions of the search space rather than exploring configurations at random. This targeted approach allows Optuna to identify optimal hyperparameter settings more efficiently than traditional methods.

2.5.2 One-Class Support Vector Machine

Support Vector Machine (SVM) was proposed by Vladimir Vapnik in 1995 as a supervised machine learning model for classification and regression tasks [15], [41]. SVM is commonly used for binary classification tasks, where a hyperplane is fitted to separate the two classes. SVM is known to be robust and requires no previous knowledge of the data or its distribution [42]. However, there are disadvantages

with SVM, such as high computational cost on large datasets, sensitivity to hyperparameter settings and performance on unbalanced datasets.

One-Class Support Vector Machine (OC-SVM) is, in contrast to SVM, an unsupervised algorithm. OC-SVM finds a hyperplane that maximizes the margin of separation to the origin [43]. Points located on the side of the hyperplane farthest from the origin are considered normal and points on the side closest to the origin are considered outliers. Schölkopf *et al.* [15] introduced OC-SVM in 1999, describing how the data is separated from the origin by solving the quadratic optimization problem:

$$\min_{w \in F, \xi \in \mathbb{R}^l, \rho \in \mathbb{R}} \quad \frac{1}{2}\|w\|^2 + \frac{1}{\nu l}\sum_i \xi_i - \rho \tag{2.2}$$

$$\text{subject to} \quad (w \cdot \phi(x_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0. \tag{2.3}$$

In this optimization problem, ν is a parameter in the range (0,1] controlling the expected proportion of outliers. The variable l denotes the number of observations and $\phi(x_i)$ is a function mapping input data to higher higher-dimensional space, often implemented using a kernel function. Slack variables are denoted by ξ_i and are introduced to relax the margin constraints and allow some data points to be misclassified, while simultaneously minimizing the extent of these misclassifications [44]. The variables w and ρ are a weight vector and an offset that define a hyperplane in the feature space.

OC-SVM is a common technique to detect outliers, but not necessarily for time series data. To use OC-SVM for time series data, a sliding window approach may be used. With this adjustment, OC-SVM interprets each window as a single data point in $d \cdot n$-dimensional space, where n is the size of the window and d denotes the number of features [45]. This transformation allows the model to classify entire segments as outliers, however, it does not consider temporal information either within or across windows. The output from the model is then a decision score s_{W_t} for each window W_t, rather than for each data point, reflecting whether the entire window is considered anomalous.

OC-SVM can be implemented with different kernel functions, which are used to transform the data to a high-dimensional space where it is linearly separable by a hyperplane [42]. Commonly used kernel functions include the Radial Basis Function (RBF) kernel and the polynomial kernel. Since real-world data is rarely linearly separable in its original space, kernel functions are essential to enable the separation between normal and anomalous data.

The RBF kernel and the polynomial kernel both have a hyperparameter γ, which defines the influence of a single training sample on the decision boundary [45]. Specifically, a lower γ value gives a smoother decision boundary, while a higher γ value gives a more complex decision boundary. The choice of γ strongly affects the model's ability to detect outliers in a general case. A too small γ may lead to underfitting, where the model is too simple to capture the underlying structure in the data, re-

sulting in poor performance on both training and unseen data [33]. In contrast, a too large γ may cause overfitting, where the model performs well on training data but fails to generalize, having effectively memorized patterns specific to the training set. This sensitivity of the hyperparameter settings is one of the main disadvantages of OC-SVM.

The parameter ν is directly connected to the fraction of Support Vectors (SVs) and the fraction of outliers [15]. Schölkopf *et al.* [15] define three properties of the ν-parameter with the following proposition:

> Assume the solution of (2.2),(2.3) satisfies $\rho \neq 0$. The following statements hold:
>
> (i) ν is an upper bound on the fraction of outliers.
>
> (ii) ν is a lower bound on the fraction of SVs.
>
> (iii) Suppose the data were generated independently from distribution P(x) which does not contain discrete components. Suppose, moreover, that the kernel is analytic and non-constant. With probability 1, asymptotically, ν equals both the fraction of SVs and the fraction of outliers. [15, p.9]

The training time of an OC-SVM model is strongly dependent on the size of the dataset [46]. A common approach to handle the dimensionality problem is to perform dimensionality reduction on the dataset. While this can improve training speed, it also drastically reduces the interpretability of the model.

2.6 Deep Learning

Deep learning is a subfield of machine learning focusing on developing and training large neural networks capable of learning complex patterns from data [33]. These models are widely used across domains like image classification, speech recognition, and medical diagnostics. As described in Section 2.5, the goal of machine learning is to approximate a function that maps inputs to outputs. In deep learning, this function is represented by an artificial neural network, a layered structure of connected units [33]. This building block is introduced in Section 2.6.1.

2.6.1 Artificial Neural Network

An artificial neural network (ANN) is a computational model inspired by the structure and function of the human brain [33]. Similar to how the brain is composed of billions of neurons that communicate by passing signals to one another, an ANN consists of many processing units called neurons that interact by exchanging information across connections.

Network Architecture

An ANN is typically structured in layers in which the neurons are organized [33]. These layers include an input layer, one or more hidden layers, and an output layer, visualized in Figure 2.2. The input layer receives raw data and passes it forward through the network. The hidden layers are responsible for processing and transforming the information by learning internal representations. Finally, the output layer produces the model's prediction or classification result. Each connection between neurons carries a numeric weight that determines how much influence a neuron's output has on the neurons in the next layer. These connections are directed, meaning information flows in one direction, as visualized with arrows in Figure 2.2. The number of neurons in each layer is referred to as the network's width, while the number of layers defines its depth [47].

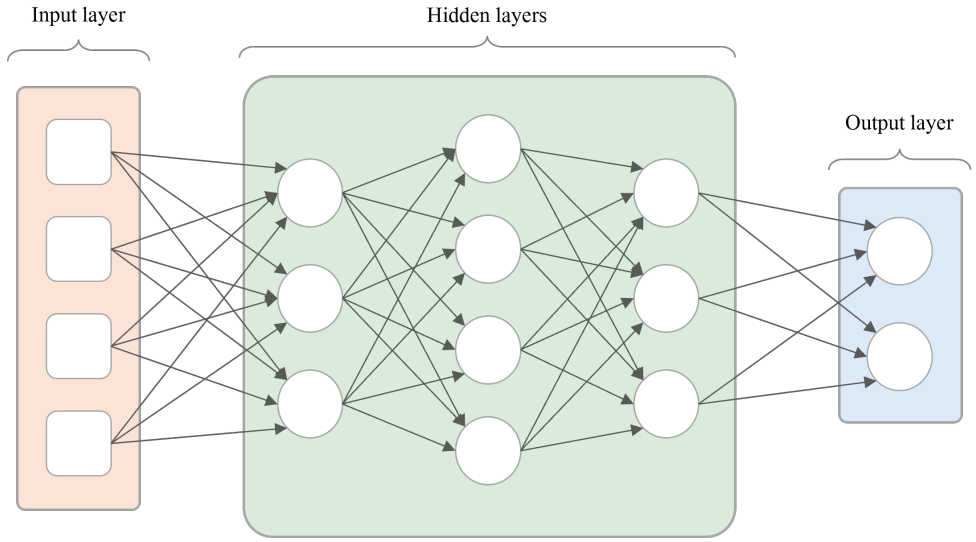


Figure 2.2: Example of an artificial neural network architecture consisting of an input layer, three hidden layers and an output layer. Each circle represents a neuron.

Figure 2.2 represents a fully connected feed-forward neural network (FNN). It is fully connected as each neuron in one layer is connected to every neuron in the next [33]. It is classified as an FNN because it contains no loops, meaning information flows only in one direction, from input to output. When the number of hidden layers is three or more, the network is referred to as a deep neural network (DNN) [48]. Deep architectures enable models to learn complex representations of data. However, increased capacity also introduces the risk of overfitting. To prevent this, dropout is a technique that randomly deactivates a subset of neurons during training to improve generalization [47].

Neuron

The neuron is the basic building block of an ANN [33]. Each neuron performs a simple two-step process to map its inputs to an output. First, it calculates a

weighted sum of its inputs. Each input x_i (with $i = 1, 2, \ldots, n$) is multiplied by a corresponding weight w_i, and the results are summed together, usually along with a bias term b. This produces an intermediate value z, expressed mathematically as:

$$z = \sum_{i=1}^{n} w_i x_i + b. \tag{2.4}$$

In the second step, z is passed through a non-linear function known as the activation function, which produces the final output of the neuron. This non-linearity is crucial as it enables the network to learn and represent complex patterns in data. Without such a function, layers in the network would behave like a linear transformation, limiting the model's ability to solve nonlinear problems. This transformation can be mathematically represented as:

$$a = \varphi(z), \tag{2.5}$$

where φ denotes the activation function applied to z. Figure 2.3 illustrates the structure of a neuron.

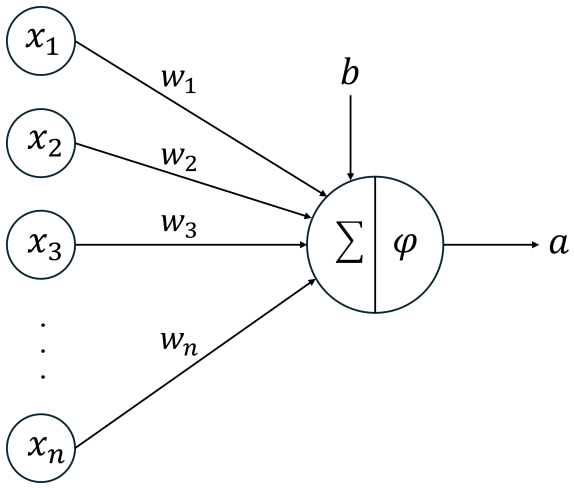
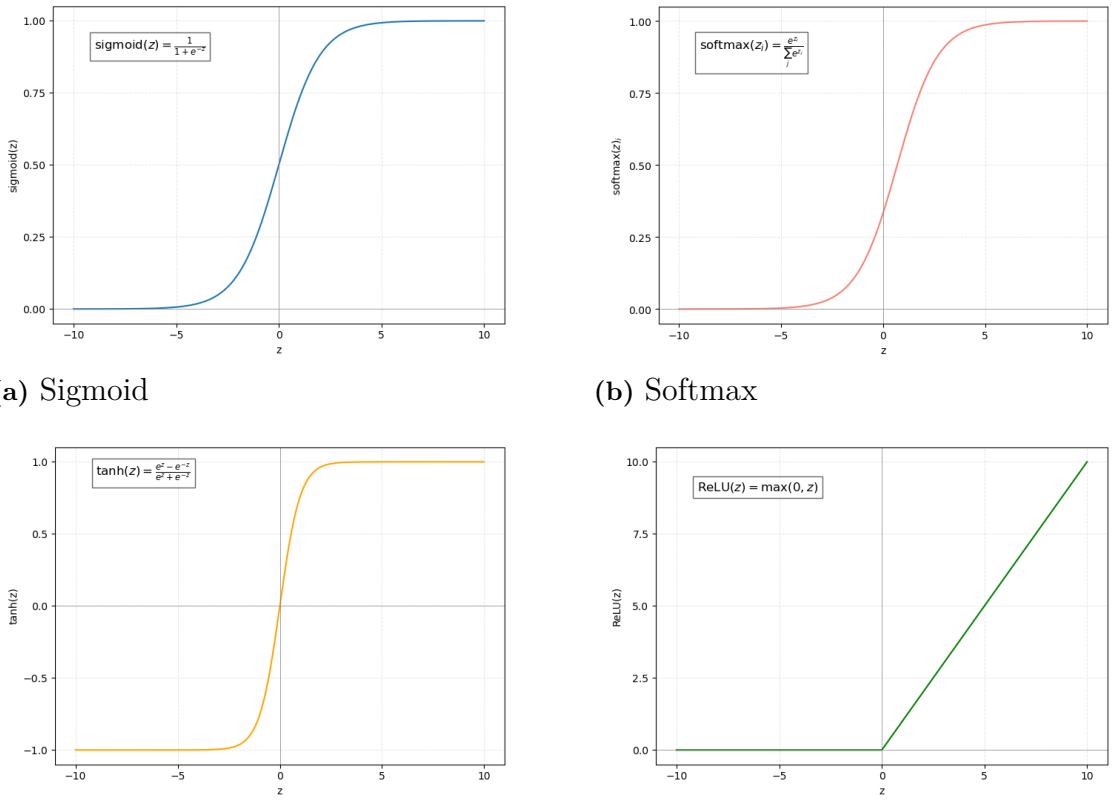


Figure 2.3: Structure of a neuron. The inputs $x_1, x_2, x_3, \ldots, x_n$ are multiplied by their corresponding weights $w_1, w_2, w_3, \ldots, w_n$, and combined with a bias term b. The summation $\sum$ represents this weighted sum plus the bias, which is then passed through an activation function φ to produce the output a of the neuron.

Activation Functions

Common activation functions include sigmoid, softmax, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU), each with different properties suited to different types of problems [33], [47]. The sigmoid function, visualized in Figure 2.4a, maps any real-valued input to the range $(0, 1)$, making it especially useful when the model output is interpreted as a probability, such as in binary classification tasks [47]. The softmax function, visualized in Figure 2.4b, is often used in the output

layer for multiclass classification tasks [47]. It converts a vector of raw scores into a probability distribution over predicted output classes, ensuring that all output values are non-negative and sum to one [47]. The tanh function, shown in Figure 2.4c, is similar in shape to the sigmoid function but scales its output to the range $(-1, 1)$. This zero-centered output allows the activations to take on both positive and negative values, which enables internal states in the network to be adjusted upward or downward during learning [33]. Moreover, Figure 2.4d shows the ReLU activation function. This function outputs zero for all negative input values and returns the input value unchanged for positive inputs. One of the key advantages of ReLU is that the function is linear in the positive domain with a derivative equal to one, allowing gradients to propagate efficiently through active units during training [33]. This property contributes to faster convergence in deep networks. However, a drawback of ReLU is that the gradient is zero for negative input values, meaning neurons operating in this region may not update their weights during training [33]. Although different neurons in a network can use different activation functions, it is common practice for all neurons within a given layer to share the same activation function [33].



(a) Sigmoid

(b) Softmax

(c) Tanh

(d) ReLU

Figure 2.4: Four standard activation functions in neural networks: sigmoid, softmax, tanh, and ReLU.

Forward and Backward Propagation

Training an ANN involves adjusting its weights to minimize the difference between the network's predicted output and the expected target values, a metric known as the loss [47]. This is achieved using the backpropagation algorithm, which operates in two main stages, a forward pass and a backward pass [33]. In the forward pass, an input is presented to the network, and activations are computed layer by layer, from the input layer through the hidden layers to the output layer. Each neuron performs its calculation and passes the result to the next layer. The weighted input z and the activation output a are stored for each neuron, which is later required in the backward pass [33]. Figure 2.5 visualizes the forward pass through a neuron.

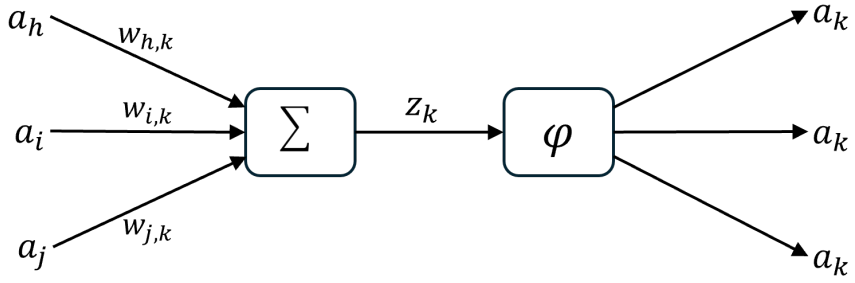


Figure 2.5: Forward propagation step for neuron k: inputs a_i, a_j, a_h are weighted by $w_{i,k}$, $w_{j,k}$, $w_{h,k}$ and summed to produce z_k, which is then passed through the activation function φ to receive the output a_k. This figure is adapted from Figure 5.2 in [49].

The backward pass begins once the network output has been produced and compared against the expected output using a loss function, such as mean squared error or cross-entropy [33]. The result is a loss value that quantifies how far the prediction deviates from the target. Backpropagation uses the loss gradient to determine how much each weight in the network contributed to the overall loss. This is done by calculating δ for each neuron, which quantifies how sensitive the loss is to small changes in that neuron's weighted input z. This loss gradient δ_k is computed as the product of two partial derivatives:

$$\delta_k = \frac{\partial \mathcal{L}}{\partial a_k} \cdot \frac{\partial a_k}{\partial z_k}. \tag{2.6}$$

Here, $\frac{\partial \mathcal{L}}{\partial a_k}$ is the partial derivative of the loss function $\mathcal{L}$ with respect to the neuron's output activation a_k, and $\frac{\partial a_k}{\partial z_k}$ is the derivative of the activation function with respect to the weighted input z_k. For output neurons, $\frac{\partial \mathcal{L}}{\partial a_k}$ is calculated directly from the loss function, typically by subtracting the target value y_k from the actual output a_k. For example, in the case of a mean squared error loss, this derivative becomes:

$$\frac{\partial \mathcal{L}}{\partial a_k} = a_k - y_k. \tag{2.7}$$

For hidden neurons, the derivative of the loss with respect to a neuron's output activation a_k is computed indirectly, using the chain rule, as a weighted sum of the loss gradients of the downstream neurons it connects to:

$$\frac{\partial \mathcal{L}}{\partial a_k} = \sum_{l \in \text{outputs}(k)} w_{k,l} \cdot \delta_l. \tag{2.8}$$

The backward pass through a hidden neuron is visualized in Figure 2.6.

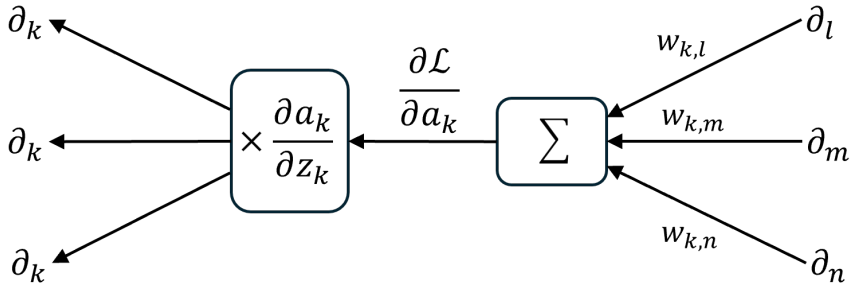


Figure 2.6: Backward propagation for hidden neuron k: the error signal $\frac{\partial \mathcal{L}}{\partial a_k}$ is backpropagated by first computing the weighted sum of gradients from subsequent neurons, then multiplied by the derivative of the activation function $\frac{\partial a_k}{\partial z_k}$ to obtain δ_k. This figure is adapted from Figure 5.3 in [49].

When the gradients δ_k have been calculated, the next step is to update the weights to reduce the overall loss. The gradient of the loss function with respect to a specific weight $w_{j,k}$, connecting neuron j to neuron k, is given by:

$$\frac{\partial \mathcal{L}}{\partial w_{j,k}} = \delta_k \cdot a_j. \tag{2.9}$$

This gradient is simply the product of the activation a_j from neuron j in the previous layer and the δ_k of the neuron k. Each weight $w_{j,k}$ is then updated using gradient descent:

$$w_{j,k}^{(\tau+1)} = w_{j,k}^{(\tau)} - \eta \cdot \frac{\partial \mathcal{L}}{\partial w_{j,k}}, \tag{2.10}$$

where η is the learning rate, and τ denote the current iteration. This process is repeated for all weights in the network and over multiple training examples until the model converges to a set of weights that minimize the loss. Different optimization algorithms can be used to update the weights during training. The most basic method is Stochastic Gradient Descent (SGD), which uses a fixed learning rate and updates the weights using an estimate of the gradient based on a small batch of examples rather than the entire dataset [47]. More advanced optimizers that adapt their learning rates during training include AdaGrad [50] and RMSProp [51].

The advantages of these two are combined in the widely used optimizer Adaptive Moment Estimation (Adam) [52], which adapts the learning rate for each parameter individually.

2.6.2 Transformer Architecture

The Transformer was introduced by Vaswani *et al.* in the paper "Attention Is All You Need" [53]. It is a kind of neural network designed to model sequential data using attention mechanisms as its core building block [53]. Unlike sequence models, such as RNNs or convolutional networks, which process data sequentially or locally, the transformer enables direct interactions between all positions in a sequence. This allows it to capture long-range dependencies more effectively while also allowing for greater parallelization [53]. The transformer architecture is visualized in Figure 2.7, and the different components of the model are described in this section.

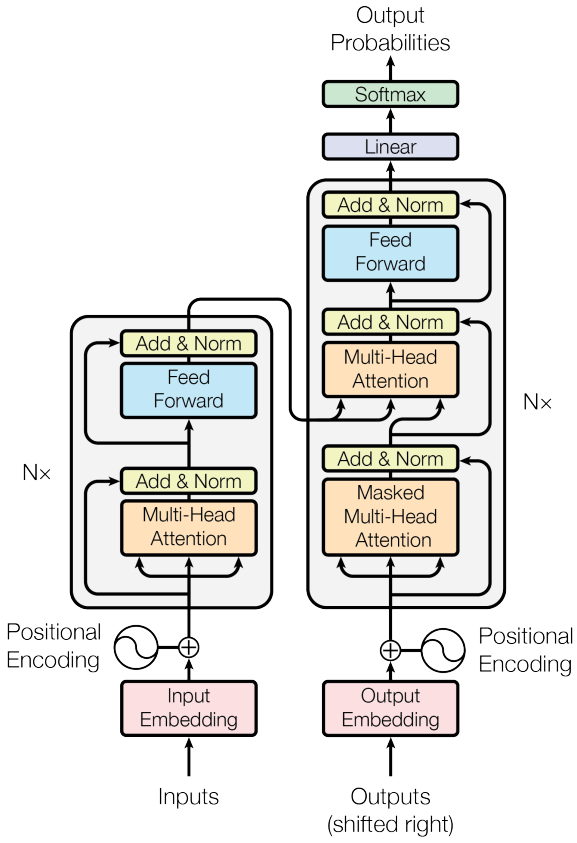


Figure 2.7: Transformer architecture, consisting of an encoder (left) and a decoder (right), each stacked N times. [54], CC-BY-SA

Encoder-Decoder Structure

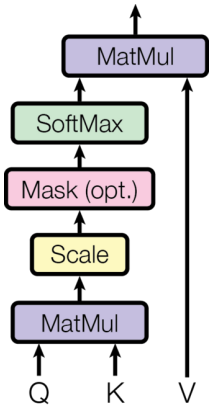
The transformer follows an encoder-decoder architecture, which is a widely used framework in sequence modeling tasks such as machine translation, speech recognition, and text generation [55]. As the name suggests, it consists of two main components, an encoder that maps the input sequence into a different space by a

projection function, and a decoder that generates the output sequence based on this encoded information [55].

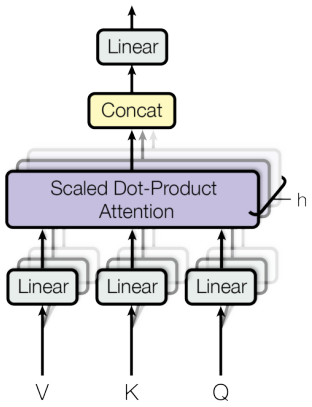
The transformer encoder, visualized in the left part of Figure 2.7, contains a stack of six identical layers, each containing two main sub-layers: a multi-head self-attention mechanism and a position-wise feed-forward neural network [53]. Each sub-layer is followed by a residual connection and layer normalization to ensure better gradient flow and training stability. The right part of Figure 2.7 illustrates the decoder, which also contains six identical layers. These layers include the same two sub-layers as in the encoder and a third sub-layer called a masked multi-head self-attention. Similar to the encoder, each sub-layer is followed by residual connections and layer normalization. The masked self-attention prevents a position from attending to future positions [53].

Attention Mechanism

The attention mechanism is the core function of the transformer, which enables the model to weigh and relate different parts of a sequence when computing representations [53]. Attention works by comparing a query to a set of keys, using the results to compute weights that determine how much focus should be placed on each corresponding value. These weighted values are then combined to produce the output.



(a) Scaled dot-product attention [56], CC-BY-SA



(b) Multi-head attention [57], CC-BY-SA

Figure 2.8: Attention mechanisms in transformers. Q, K, and V denote the query, key, and value vectors, respectively.

The type of attention used in the transformer is known as scaled dot-product attention, illustrated in Figure 2.8a. Given a matrix of queries Q, keys K, and values V, the attention output is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \tag{2.11}$$

Here, the dot product $QK^\top$ measures the similarity between each query and key. Dividing by $\sqrt{d_k}$, where d_k is the dimension of the key vectors, scales the result to avoid extremely large values that could push the softmax function into regions with very small gradients. The output is a weighted sum of the values V, where the weights are determined by this scaled similarity.

To improve the model's ability to capture different patterns in the data, the transformer uses multi-head attention, shown in Figure 2.8b. Instead of performing a single attention operation, the inputs are linearly projected h times, and attention is computed in parallel. The results are concatenated and linearly transformed to produce the final output, mathematically expressed:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(H_1, ..., H_h)\mathbf{w}^O, \tag{2.12}$$

$$\text{where } H_i = \text{Attention}(Q\mathbf{w}_i^Q, K\mathbf{w}_i^K, V\mathbf{w}_i^V). \tag{2.13}$$

Here, $\mathbf{w}_i^Q, \mathbf{w}_i^K$, and $\mathbf{w}_i^V$ are learned projection matrices for each attention head, and $\mathbf{w}^O$ is the output projection matrix that maps the concatenated outputs back to the model dimension. In the original transformer architecture, $h = 8$ and each head H_i processes a lower-dimensional version of the input.

Position-Wise Feed-Forward Neural Network

Each layer in both the encoder and decoder includes a fully connected feed-forward neural network, applied after attention [53]. This component operates independently on each position in the sequence, using the same parameters across all positions within a layer. The FNN consists of two linear layers separated by a ReLU activation, mathematically defined as:

$$\text{FNN}(x) = \max(0, x\mathbf{w}_1 + b_1)\mathbf{w}_2 + b_2. \tag{2.14}$$

Here, x is the input vector at a particular position in the sequence, typically the output of the attention sub-layer. The matrix $\mathbf{w}_1$ contains learned weights that project the input to a higher-dimensional space, followed by a ReLU activation to introduce non-linearity. The second matrix $\mathbf{w}_2$ then projects the result back to the original space. The bias terms b_1 and b_2 are added at their respective step. As the FNN is applied separately and identically to each position, it encodes the local context at that position into a higher-level representation [58].

Positional Encoding

As described earlier, the transformer has no recurrence and no convolution, meaning that the order of the sequence is not inherently accounted for unless it is introduced explicitly [53]. In the transformer, this information is provided through positional encodings, which are added to the input embeddings before they are passed into the encoder and decoder. These encodings are based on sine and cosine functions

of varying frequencies. Their purpose is to inject information about the position of elements in the sequence, allowing the model to take sequence order into account despite its non-sequential architecture.

2.6.3 TranAD

TranAD is a deep transformer network developed specifically for anomaly detection in time series data [23]. Similar to a conventional transformer model, TranAD follows an encoder-decoder structure with two encoders and two decoders, as can be seen in Figure 2.9. A problem when using conventional transformers for anomaly detection is that subtle anomalous segments yield small reconstruction loss and could therefore be missed [23], [59]. In the case of transformer models, loss, or reconstruction loss, refers to the difference between the predicted output and the expected target. For TranAD, Tuli *et al.* [23] propose that a solution is to use an adversarial training process to amplify reconstruction errors. The model is trained in two phases, with the first phase dedicated to learning to reconstruct the input windows. The reconstruction loss from the first decoder from phase 1 is then used as the focus score in the second phase, allowing the model to focus more on subsequences with higher loss.

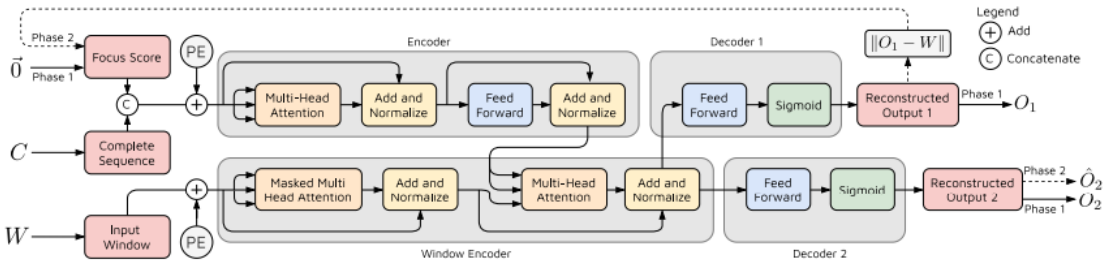


Figure 2.9: Overview of the TranAD architecture. $\vec{0}$ denotes the zero vector, C denotes context and W denotes the input window. PE refers to positional encoding. [23], CC-BY-NC-ND

TranAD consists of two encoders and two decoders. The output from the two decoders are reconstructions of the input window W and are denoted as O_1 and O_2 for phase 1 and $\hat{O}_2$ for phase 2. The output from decoder 1 is not used in phase 2 and can therefore be ignored.

Context Encoder

The context encoder generates an encoding of a bigger contextual window until the current timestamp using focus scores. In phase 1, the focus score is a zero matrix with the same dimensions as the input window. In phase 2, the focus score is the reconstruction loss from decoder 1. The purpose of using a focus score is to draw focus to parts of the input window with the highest reconstruction loss, working as a prior to modify the attention weights. The authors of the model state three

benefits of this approach: a simplified labeling process, false positive prevention and an improved generalization due to adversarial style training [23].

The following operations are done in the context encoder:

$$\begin{aligned} I_1^1 &= \text{LayerNorm}(I_1 + \text{MultiHeadAtt}(I_1, I_1, I_1)), \\ I_2^1 &= \text{LayerNorm}(I_1^1 + \text{FeedForward}(I_1^1)). \end{aligned} \tag{2.15}$$

I_1 refers to the concatenation of all windows up until the current timestamp and the corresponding focus scores. The architecture of the encoder can also be seen in Figure 2.9, where the context encoder is denoted as encoder.

Window Encoder

The window encoder includes the encoding of the context until the current timestamp, given by the context encoder, to create an encoding of the input window [23]. The encoding of the input window is then passed on to the two decoders.

The following operations are done in the window encoder:

$$\begin{aligned} I_1^2 &= \text{Mask}(\text{MultiHeadAtt}(I_2, I_2, I_2)), \\ I_2^2 &= \text{LayerNorm}(I_2 + I_1^2), \\ I_3^2 &= \text{LayerNorm}(I_2^2 + \text{MultiHeadAtt}(I_2^1, I_2^1, I_2^2)). \end{aligned} \tag{2.16}$$

The overall architecture of the window encoder is similar to that of the context encoder. In contrast, the input to the window encoder, I_2, is a single window to which masked attention is applied to prevent leaking future information.

Decoders

The model consists of two identical decoders that generate two reconstructions of the input window. The reconstruction loss from the first decoder in the first phase is used as a focus score for the context encoder in the second phase. The following operations are done in the two decoders:

$$O_i = \text{Sigmoid}(\text{FeedForward}(I_3^2)), \quad i \in \{1, 2\}. \tag{2.17}$$

To align with the normalized input window, the Sigmoid activation function is used to generate an output in the range [0,1].

Loss

In TranAD, the loss, or reconstruction loss, for each decoder is defined using the L2-norm [23]:

$$\begin{aligned} \mathcal{L}_{\text{Decoder}_1} &= \|O_1 - W\|_2, \\ \mathcal{L}_{\text{Decoder}_2} &= \|O_2 - W\|_2. \end{aligned} \tag{2.18}$$

To make the model more robust and to improve its generalizability, TranAD employs an adversarial loss mechanism, using the outputs from the second phase [23]. The reconstruction loss from decoder 1 in the first phase is used as focus scores in the second phase. These scores indicate which parts of the input window were poorly reconstructed and thus potentially anomalous. In an adversarial fashion, the two decoders play opposing roles. Decoder 1 tries to minimize the difference between the reconstruction and the target, effectively trying to recreate the input window. Decoder 2, in contrast, tries to maximize the difference between the reconstruction and the target by identifying areas where the reconstruction is poor. This can be expressed as:

$$\min_{\text{Decoder}_1} \max_{\text{Decoder}_2} \|\hat{O}_2 - W\|_2. \tag{2.19}$$

The aim is to force decoder 1 to create more accurate reconstructions, while decoder 2 learns to amplify errors that might indicate anomalies. This can also be expressed in terms of loss:

$$\begin{aligned} \mathcal{L}_{\text{Decoder}_1} &= +\|\hat{O}_2 - W\|_2, \\ \mathcal{L}_{\text{Decoder}_2} &= -\|\hat{O}_2 - W\|_2, \end{aligned} \tag{2.20}$$

where $\mathcal{L}_1$ and $\mathcal{L}_2$ are the losses for decoder 1 and decoder 2, respectively. The reconstruction loss for the decoders from both phases is then combined in a cumulative, evolutionary loss function for each decoder. To ensure stable training, focus gradually shifts from standard reconstruction loss to adversarial loss over time [23]. This evolutionary loss is expressed as:

$$\begin{aligned} \mathcal{L}_{\text{Decoder}_1} &= \epsilon^{-n}\|O_1 - W\|_2 + (1 - \epsilon^{-n})\|\hat{O}_2 - W\|_2, \\ \mathcal{L}_{\text{Decoder}_2} &= \epsilon^{-n}\|O_2 - W\|_2 - (1 - \epsilon^{-n})\|\hat{O}_2 - W\|_2, \end{aligned} \tag{2.21}$$

where ϵ is a constant close to one and n is the current epoch. Over time, the weight of the first term decreases and the weight of the second term increases. This is done in order to prioritize reconstruction loss at early stages and then shift the weight to adversarial loss as reconstructions get better. As reconstructions tend to be poor at early stages, they can be unreliable as priors for the adversarial loss and thus destabilize the training process.

During testing, anomaly scores for unseen input windows, $\hat{W}$, are computed similarly to the evolutionary loss expressed in equation 2.21:

$$s_{\hat{W}_t} = \frac{1}{2}\|O_1 - \hat{W}_t\|_2 + \frac{1}{2}\|\hat{O}_2 - \hat{W}_t\|_2. \tag{2.22}$$

Here, $s_{\hat{W}_t}$ is a single anomaly score assigned to input window $\hat{W}_t$. Once the anomaly scores have been computed for all input windows and all dimensions, a window can be labeled as anomalous if it exceeds some threshold.

Peaks-Over-Threshold

A common approach in anomaly detection is to use Peak-Over-Threshold (POT) to dynamically set this threshold [23]. POT is a statistical approach based on Extreme Value Theory that fits the tail of the anomaly scores to a Generalized Pareto Distribution (GPD) [60].

More specifically, given a sequence of anomaly scores $\{s_{i-N+1}, \ldots, s_i\}$, POT first selects a high threshold r, using a chosen percentile p, and focuses only on the scores exceeding r. These excesses, defined as $e_i = s_i - r$ for each $s_i > r$, are assumed to follow a GDP with shape parameter ξ and a scale parameter σ. To estimate these parameters, maximum likelihood estimation is typically used. Once fitted, POT uses the GPD parameters to compute an adjusted anomaly score z_q corresponding to a specified risk level q. This threshold is set such that the probability of an anomaly score s_i exceeding z_q is less than q, formally:

$$P(s_i > z_q) < q. \tag{2.23}$$

The threshold z_q is calculated as:

$$z_q = r + \frac{\sigma}{\xi}\left(\left(\frac{qn}{n_r}\right)^{-\xi} - 1\right), \tag{2.24}$$

where n is the number of observations and n_r is the number of peaks observed above r. A smaller q leads to a higher threshold, resulting in fewer anomalies being flagged, while a larger q lowers the threshold, making the detector more sensitive.

2.7 Performance Metrics

In supervised settings, or if a labeled testset is available, it is common practice to visualize the results of the model in a confusion matrix and to compute metrics such as F1 score, accuracy, precision, recall and ROC/AUC [61]. A binary confusion matrix is used to visualize classification results [62]. As shown in Figure 2.10, the matrix consists of four boxes, representing true positive (TP), false negative (FN), false positive (FP) and true negative (TN). These classifications are used to compute different metrics to evaluate the performance of a model.

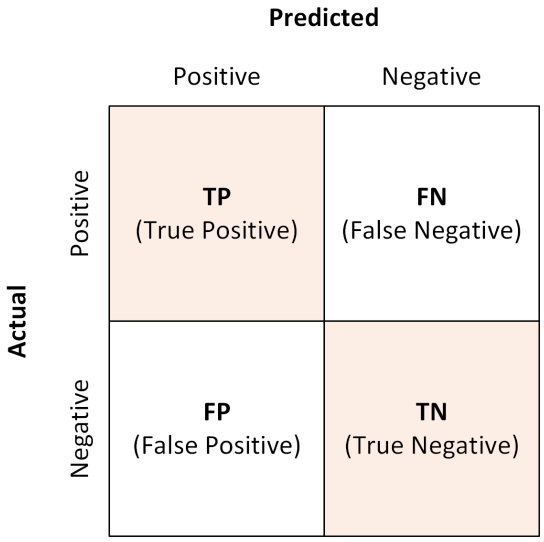


Figure 2.10: Illustration of a confusion matrix for a binary classification task. The matrix includes true positives, false positives, true negatives and false negatives.

2.7.1 Accuracy

Accuracy is the proportion of correct classifications, both positive and negative [61], and is mathematically defined as:

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{all classifications}} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.25}$$

Accuracy needs to be used with care, as it is dependent on the balance of the dataset and can be misleading if used on an imbalanced dataset. For a dataset with few negatives and many positives, a model that predicts all positives would achieve a good accuracy even though all negatives are misclassified.

2.7.2 Precision

Precision is the proportion of positive classifications that are correct [63]. Mathematically defined as:

$$\text{Precision} = \frac{\text{correctly predicted positives}}{\text{everything classified as positive}} = \frac{TP}{TP + FP}. \tag{2.26}$$

When applied to an imbalanced dataset with a large proportion of negative instances, precision becomes less relevant as a performance metric. While it measures how many of the predicted positive cases are correct, it does not indicate how effectively the model identifies all actual positive cases.

2.7.3 Recall

Recall, also known as true positive rate or sensitivity, is the proportion of positives being correctly classified as positives [63]. Mathematically defined as:

$$\text{Recall} = \frac{\text{correctly predicted positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}. \tag{2.27}$$

Similar to precision, recall can be misleading when used on an imbalanced dataset.

2.7.4 F1 Score

Precision and recall can both be misleading and neither is optimal to use on its own [63]. As a result, F1 score has become a common choice to combine precision and recall scores into one metric. Mathematically defined as:

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2TP}{2TP + FP + FN}. \tag{2.28}$$

An F1 score close to 1 means the model has achieved a good combined precision and recall score.

2.7.5 ROC

While metrics like precision, recall, and F1 summarize the model performance at a single threshold, the Receiver Operating Characteristic (ROC) curve gives a more complete evaluation of the classifier's performance by considering all possible thresholds [64], described later in this section. It shows how well the model separates positive and negative instances, specifically illustrating the trade-off between detecting TP and allowing FP.

The ROC curve is plotted within a two-dimensional graph (ROC space) [64], illustrated in Figure 2.11. The x-axis represents the False Positive Rate (FPR) and the y-axis represents the True Positive Rate (TPR). These rates describe how often the classifier makes errors or correct decisions relative to the actual class labels. They are mathematically defined as:

$$FPR = \frac{\text{incorrectly predicted negatives}}{\text{all actual negatives}} = \frac{FP}{FP + TN}, \tag{2.29}$$

$$TPR = \frac{\text{correctly predicted positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}. \tag{2.30}$$

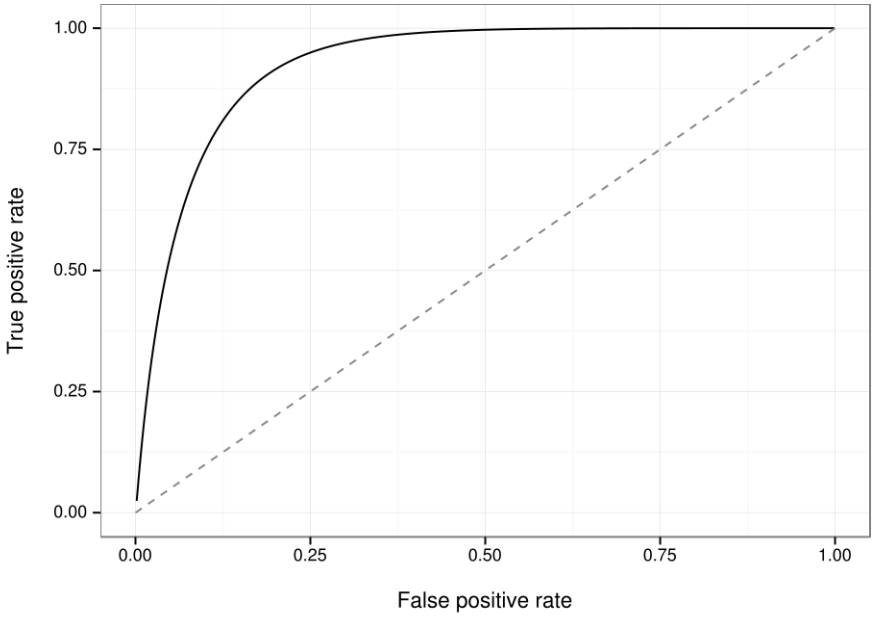


Figure 2.11: Example of a ROC curve showing the true positive rate versus the false positive rate [65], CC-BY-SA

Some classifiers, such as rule-based systems, produce one single class label for each instance, typically positive or negative [64]. When evaluated on a test set, such classifiers result in a single confusion matrix and correspond to one point in the ROC space. However, many classifiers output a score that reflects how likely an instance is predicted to belong to the positive class. These scores can be probabilities or confidence values, as long as higher scores indicate greater belief in the positive class [64]. In the context of this thesis, such scores refer to the previously described anomaly scores, which quantify how anomalous each window is.

To turn these scores into binary predictions, a decision threshold is applied [64]. Instances with scores above the threshold are classified as positive, and the rest as negative. By varying this threshold from high to low, different combinations of TPR and FPR are obtained, each corresponding to a different point in ROC space. Connecting these points forms the ROC curve [64]. The curve's shape shows how well the model ranks positive instances above negative ones [64]. A curve that approaches the top-left corner $(0, 1)$ indicates strong performance, high TPR with low FPR. A curve close to the diagonal suggests performance no better than random guessing.

2.7.6 AUC

The Area Under the Curve (AUC) corresponds to the area under the ROC curve and is thus represented as a single number between 0 and 1 [61]. A high AUC score indicates a good model, with a score of 1 representing perfect separation between positive and negative instances. The diagonal dashed line in Figure 2.11 represents an AUC score of 0.5.

2.7.7 Range-Based Metrics

The limitation of classical point-wise evaluation metrics, such as those previously discussed, is their inability to account for the temporal structure of anomalies. As described in Section 2.3, this thesis focuses on collective anomalies, which are patterns that occur over a range of consecutive timesteps rather than at single points. Evaluating how well a model can detect such anomalies requires metrics that consider not just point-wise accuracy, but segment-wise detection. To address this, a recent approach introduces range-based precision (P_X) and range-based recall (R_X) [66]. These metrics account for partial overlaps, detection fragmentation, and positional bias when comparing predicted and ground truth anomaly windows.

Range-Based Precision

The range-based precision evaluates how well predicted anomaly ranges align with ground truth anomaly windows [66]. Formally, let $\mathcal{R} = \{R_1, R_2, ..., R_{N_r}\}$ be the set of ground truth anomaly ranges and $\mathcal{P} = \{P_1, P_2, ..., P_{N_p}\}$ the set of predicted anomaly ranges. The range-based precision is defined as the average precision over all predicted ranges:

$$P_X(\mathcal{R}, \mathcal{P}) = \frac{1}{N_p} \sum_{i=1}^{N_p} P_X(\mathcal{R}, P_i), \tag{2.31}$$

where N_p is the number of predicted anomaly ranges. For each predicted range $P_i \in \mathcal{P}$, an individual precision score $P_X(\mathcal{R}, P_i)$ is computed based on how well it overlaps with the ground truth anomaly ranges in $\mathcal{R}$:

$$P_X(\mathcal{R}, P_i) = \text{CardinalityFactor}(P_i, \mathcal{R}) \cdot \sum_{j=1}^{N_r} \text{OverlapSize}(P_i, R_j, \text{PositionalBias}). \tag{2.32}$$

In this formulation, the cardinality factor function penalizes fragmented predictions by reducing the score when P_i overlaps multiple distinct ground truth anomaly ranges [66]. A common approach is to define the penalty as inversely proportional to the number of ground truth anomaly ranges that P_i intersects. The cardinality factor is then defined as:

$$\text{CardinalityFactor}(P_i, \mathcal{R}) = \frac{1}{|\{R_j \in \mathcal{R} \mid P_i \cap R_j \neq \emptyset\}|}. \tag{2.33}$$

The second component in Equation 2.32 is the overlap size function. This function measures the weighted extent of the overlap between P_i and each ground truth anomaly range R_j [66]. This weighting can be adjusted through a positional bias function to reflect application-specific importance of early or late detection, denoted

as 'PositionalBias'. Assuming a flat positional bias, where all positions are considered equally important, the overlap size function is defined as:

$$\text{OverlapSize}(P_i, R_j) = \frac{|P_i \cap R_j|}{|P_i|}. \tag{2.34}$$

Range-Based Recall

While range-based precision focuses on the quality of predicted anomaly ranges, range-based recall evaluates how well the model captures the ground truth anomaly ranges [66]. It measures how much of each ground truth anomaly was successfully detected by the predictions. The overall recall is defined as the average recall across all ground truth anomaly ranges:

$$R_X(\mathcal{R}, \mathcal{P}) = \frac{1}{N_r} \sum_{i=1}^{N_r} R_X(R_i, \mathcal{P}), \tag{2.35}$$

where N_r is the number of ground truth anomaly ranges. The recall for an individual ground truth anomaly range R_i is computed using two components: an existence reward, indicating whether the anomaly was detected at all, and an overlap reward, reflecting the extent and quality of the detected region. These are weighted by parameters α and β, respectively, where $\alpha + \beta = 1$:

$$R_X(R_i, \mathcal{P}) = \alpha \cdot \text{ExistenceReward}(R_i, \mathcal{P}) + \beta \cdot \text{OverlapReward}(R_i, \mathcal{P}). \tag{2.36}$$

The existence reward is defined as:

$$\text{ExistenceReward}(R_i, \mathcal{P}) = \begin{cases} 1, & \text{if } \exists P_j \in \mathcal{P} \text{ such that } P_j \cap R_i \neq \emptyset \\ 0, & \text{otherwise} \end{cases}. \tag{2.37}$$

The overlap reward measures the weighted extent of alignment between R_i and the predicted ranges:

$$\begin{aligned} \text{OverlapReward}(R_i, \mathcal{P}) = \text{CardinalityFactor}(R_i, \mathcal{P}) \cdot \\ \sum_{j=1}^{N_p} \text{OverlapSize}(R_i, P_j, \text{PositionalBias}). \end{aligned} \tag{2.38}$$

In contrast to the precision formulation, the cardinality factor here penalizes fragmented detections, meaning that multiple distinct predictions identify a single ground truth anomaly.

$$\text{CardinalityFactor}(R_i, \mathcal{P}) = \frac{1}{|\{P_j \in \mathcal{P} \mid R_i \cap P_j \neq \emptyset\}|}. \tag{2.39}$$

Similarly, the overlap size function is normalized with respect to the ground truth anomaly R_i, not the prediction. Assuming a flat positional bias, it is defined as:

$$\text{OverlapSize}(R_i, P_j) = \frac{|R_i \cap P_j|}{|R_i|}. \tag{2.40}$$

Figure 2.12 provides a visual summary of how range-based metrics evaluate predictions. It illustrates the effects of multiple matching predictions and partial overlaps. Together, these examples clarify how cardinality, overlap size, and existence reward are interpreted.

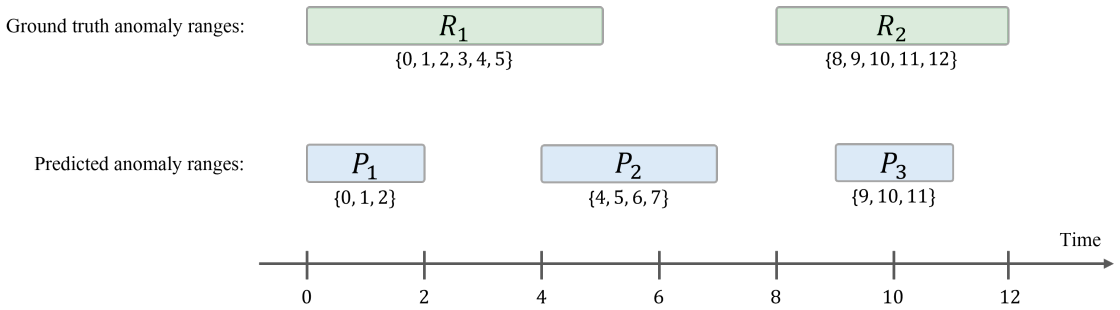


Figure 2.12: Illustration of range-based evaluation components for anomaly detection. Two ground truth anomaly ranges, R_1 and R_2, are shown in green, and three predicted ranges, P_1, P_2, and P_3, in blue. R_1 overlaps partially with both P_1 and P_2, resulting in an existence reward of 1 and a cardinality factor of 1/2. The normalized overlap size is computed for each prediction or ground truth, depending on whether precision or recall is being measured. R_2 is partially detected by a single prediction, P_3, resulting in an existence reward of 1 and a cardinality factor of 1. The overlap between R_2 and P_3 is partial: from the recall perspective, P_3 captures 3/5 of the ground truth range, while from the precision perspective, the entire predicted range lies within the ground truth, yielding an overlap size of 1.

2.7.8 IoU

Intersection over Union (IoU) is a standard metric for evaluating the accuracy of predicted regions against ground truth, particularly in detection tasks [67]. In the context of this work, IoU is used to measure the overlap between predicted anomalous segments and the true anomalous segments in the time series. Mathematically, given a ground truth anomaly range R_j and a predicted anomaly range P_i, the IoU is defined as:

$$\text{IoU}(P_i, R_j) = \frac{|P_i \cap R_j|}{|P_i \cup R_j|}. \tag{2.41}$$

3

Data Description

The available data consist of high-frequency time series data collected from two main sources: field test vehicles and test cells. While the datasets are similar, they differ significantly in terms of size. The field test dataset contains more than 500,000,000 rows, whereas the test cell dataset contains around 500,000 rows. The signals used in this thesis, excluding metadata, can be seen in Table 3.1. Sections 3.1 and 3.2 describe the field test and test cell data, respectively. Section 3.3 covers the manual data recordings, while Section 3.4 details the manually discovered oscillations.

Table 3.1: Overview of the vehicle signals used in this thesis, including their corresponding units and data types.

Feature	Unit	Datatype
Engine speed	rpm	float
Engine torque	Nm	float
Actuator signal	%	float

3.1 Field Test Data

The field test data is logged by test vehicles, via CAN or other network connections, continuously as the truck is active. These logs capture signals from various control units, usually at a frequency of 10 Hz, but in some cases reaching up to 100 Hz. On average, a field test vehicle logs from 500 to 1000 signals. Trucks are active for different periods of time, ranging from a few hours a day to 24 hours a day. Since the data is generated under real-world driving conditions, it is subject to a range of external factors, such as driver behavior and traffic conditions, that introduce variability in the data.

3.2 Test Cell Data

The test cell data is logged at a frequency of 10 Hz during simulated driving scenarios. Unlike field test data, which is collected from active trucks, test cell data is generated in a controlled environment known as a test cell. As there are no external factors affecting the data, it is generally less prone to oscillations. However, the

extent of oscillatory behavior varies between different tests, with some being more prone to oscillations than others. An example of an oscillation found in the test cell data is shown in Figure 3.1.

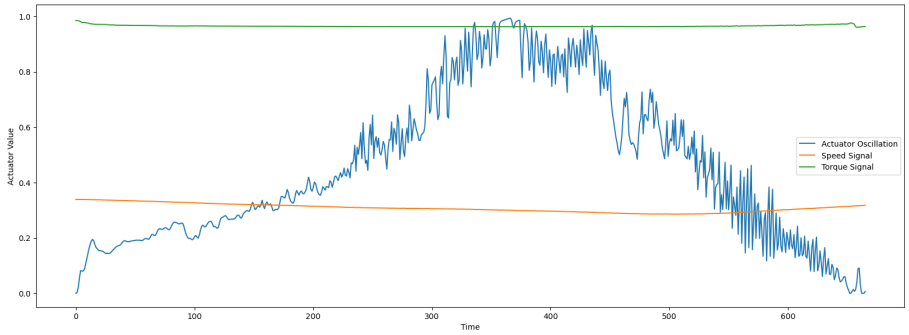


Figure 3.1: Example of an oscillation found in the test cell data. The blue line shows the oscillating actuator, while the green line shows the engine torque and the orange line shows the engine speed.

3.3 Manual Data Recordings

Apart from the field test and test cell datasets, there are also 13 manual data recordings of oscillations. The sampling rate of each recording differs slightly, typically at a frequency of 70-80 Hz. A manual data recording is done by an individual who directly connects to a CAN bus in a truck.

3.4 Manually Discovered Oscillations

In total, 36 oscillations were extracted, 13 coming from manual data recordings and 23 from test cell data. The manual recordings, provided by domain experts at Volvo Group, contain previously identified oscillations. To supplement these, several test cell files with known oscillations were also provided. Based on information from domain experts, these files were inspected to manually identify additional examples. Intervals with oscillatory behavior were extracted and saved in CSV format. Instances where both the actuator and torque signals displayed oscillatory behavior were excluded, as actuator oscillations in these cases are usually a response to the torque oscillations. While torque oscillations might be anomalous, the focus of this thesis is oscillations in the actuator signal. An example of a manually discovered and extracted oscillation from the test cell can be seen in Figure 3.1.

4

Methods

In this chapter, the methodology used to detect oscillatory behavior in time series data is presented. It begins by outlining the data preprocessing steps in Section 4.1. Thereafter, the data sets used for training and testing are described in Section 4.2. Section 4.3 introduces the implementation of OC-SVM and the TranAD model. Finally, Section 4.4 explains the evaluation metrics and experimental setup used to validate the performance of the methods.

4.1 Data Preprocessing

The first stage of the method involves preparing the raw input data for analysis through preprocessing steps. These include downsampling, selecting relevant signals and normalizing values.

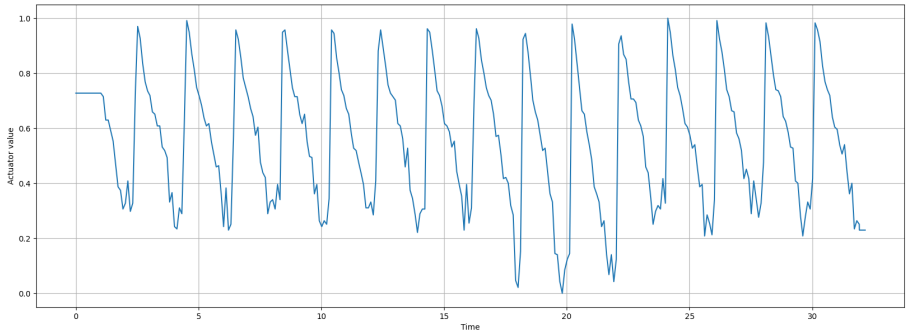
4.1.1 Downsampling

The available data described in Chapter 3 were logged with varying sampling frequencies, with 10 Hz being the lowest. To ensure consistency across datasets, all data were downsampled to 10 Hz. While downsampling unavoidably results in some loss of information, this trade-off was deemed necessary.

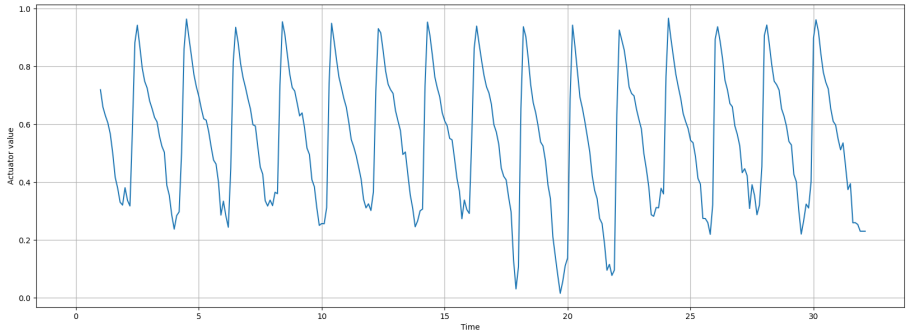
The field test data was downsampled by using an internal Volvo Group data preprocessing API, where the last value in each 100 ms (10 Hz) interval was retained. However, this API was not compatible with the dataset of manual recordings, so a similar function was developed for this data. This function replicated the behavior of the API function by dividing the data into bins of 100 ms and retaining the last value in each bin. Figure 4.1 shows an example of data downsampled from 80 Hz to 10 Hz.

4.1.2 Signal Selection

One actuator was identified by domain experts at Volvo Group as a key point of interest. Oscillations have been manually detected in this actuator, but the extent to which they occur is unknown. Engine speed and engine torque were included in the dataset to provide some context to the actuator.



(a) Oscillatory pattern in manual data recording with frequency of 80 Hz



(b) The same data, but downsampled to 10 Hz

Figure 4.1: Example of a manual data recording of an oscillation in an actuator in 80 Hz. The data was downsampled to 10 Hz, preserving the overall pattern despite some minor loss of detail.

4.1.3 Normalization

All data was rescaled to the range [0,1] using min-max normalization. This ensures that each signal contributes equally, preventing features with larger scales from being disproportionately weighted.

4.2 Data Sets

This section describes the data sets used for training and testing. Due to the limited availability of manually discovered oscillations, synthetic oscillatory segments were generated to provide additional test data and support broader evaluation. The labeling process and further details of the testing data are presented in Section 4.2.3.

4.2.1 Training Sets

Three training sets were used in this thesis, one for OC-SVM, one for TranAD and one shared between both. The available test cell data was limited, as can be seen in 4.1, whereas the field test data was considerably more extensive. Due to

the computational demands of OC-SVM, the field test training set was limited to 1,000,000 rows for this model.

The test cell training set was created by selecting data from tests with minimal oscillatory behavior, based on guidance from domain experts. The goal was to construct a clean training set that accurately represents normal behavior, so that oscillations would be more likely to stand out as anomalies during detection.

Table 4.1: Overview of the training sets, including their corresponding data sources, sizes and the models they were used for.

Data Source	Number of rows	Model
Field test	1,000,000	OC-SVM
Field test	100,000,000	TranAD
Test cell	500,000	OC-SVM & TranAD

4.2.2 Synthetic Data Generation

To simulate a wide range of realistic oscillatory signals, a synthetic data generation framework was developed. It generates time series by starting with a simple sine or cosine waveform, which is then modulated with an amplitude variation function and optionally shifted using a trend function. This allows the simulation of a wide range of oscillatory patterns with different dynamics. The following subsections describe each of these components in detail.

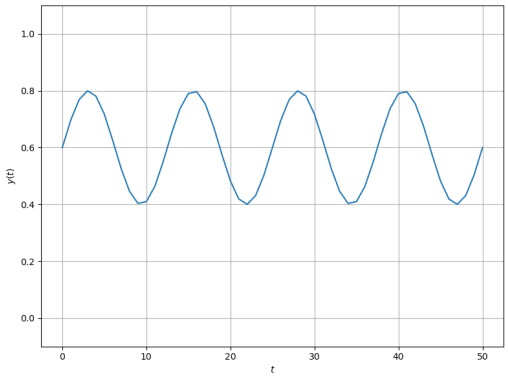
Base Waves

A base wave is generated using standard sine and cosine functions with normalization. These waves are produced as:

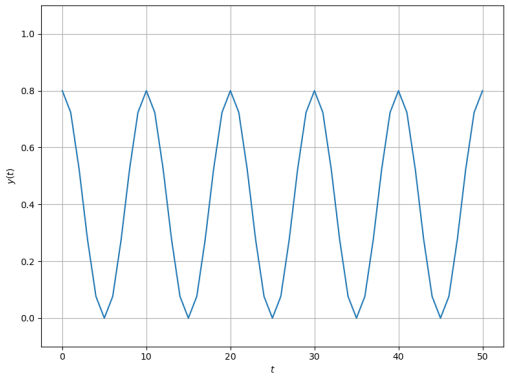
$$y_{\text{sine}}(t) = a \cdot \frac{\sin(2\pi f t) + 1}{2}, \tag{4.1}$$

$$y_{\text{cosine}}(t) = a \cdot \frac{\cos(2\pi f t) + 1}{2}, \tag{4.2}$$

where $a \in (0, 1]$ is the base amplitude of the time series, f is the frequency and t is the time vector. This mapping transforms the standard sine function, which oscillates between -1 and 1, into a waveform normalized to the interval $[0, 1]$. The frequencies of the synthetic oscillations are selected by manually analyzing and matching observed real oscillations. A vertical offset v is also added in a later step to shift the entire waveform up or down along the vertical axis. This enables the signal to occupy different regions of the amplitude space while retaining its oscillatory shape. Figure 4.2 shows two samples of the sine and cosine base waves, with Figure 4.2a including a vertical shift of 0.4, and Figure 4.2b showing a wave without vertical offset.



(a) Constant sine wave generated with function (4.1) with $a = 0.4$, $f = 0.08$, and $v = 0.4$.



(b) Constant cosine wave generated with function (4.2) with $a = 0.8$, $f = 0.1$, and $v = 0$.

Figure 4.2: Two synthetic time series samples of constant sine and cosine waves, of lengths 50.

Amplitude Variation Functions

To vary the amplitude of the base waves, and simulate non-stationary behavior to match real oscillations, different functions $A(t)$ are applied to the final synthetic time series according to the equation:

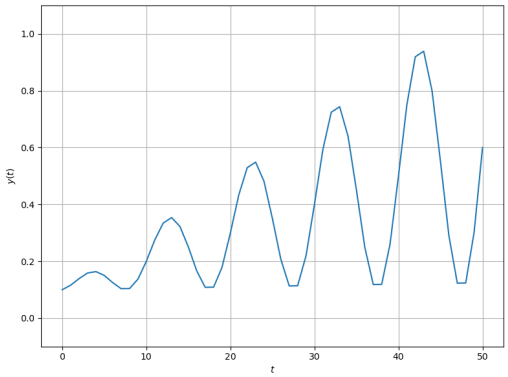
$$y_{\text{modified}}(t) = y_{\text{base}}(t) \cdot A(t) + v, \tag{4.3}$$

where $y_{\text{modified}}(t)$ is the modified time series, y_{base} is the base wave, and v is the vertical offset used to shift the waveform along the vertical axis. Linear variations, including increase and decrease, are applied with the following functions:

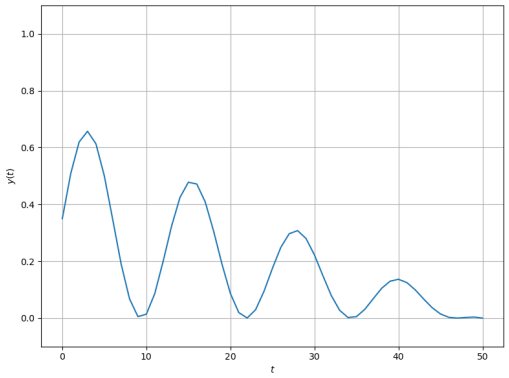
$$A_{\text{linear\_increase}}(t) = \frac{t}{T}, \tag{4.4}$$

$$A_{\text{linear\_decrease}}(t) = 1 - \frac{t}{T}, \tag{4.5}$$

where t is the relative time within the synthetic oscillatory segment and T denotes the length of the time series. Figure 4.3 shows two examples of time series with sine waves and linear increase and decrease amplitude variation functions.



(a) Sine wave with the linear increase function (4.4) applied, with $a = 1.0$, $f = 0.1$ and $v = 0.1$.



(b) Sine wave with the linear decrease function (4.5) applied, with $a = 0.7$, $f = 0.08$, and $v = 0$.

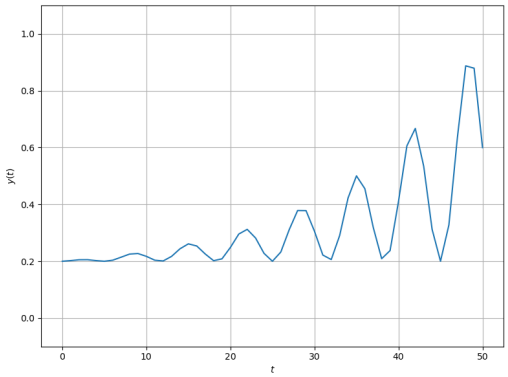
Figure 4.3: Two time series samples of sine waves with linear increase and decrease amplitude variation functions applied. Both time series are of length 50.

Exponential increases and decreases were simulated to represent scenarios such as rapid signal growth or damping. The following functions were then integrated:

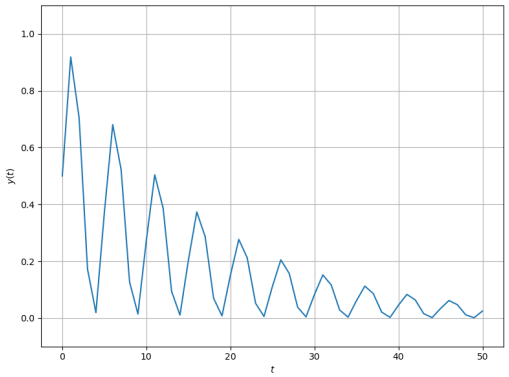
$$A_{\text{exponential\_increase}}(t) = \frac{e^{3t/T} - 1}{e^3 - 1}, \tag{4.6}$$

$$A_{\text{exponential\_decrease}}(t) = e^{-\frac{3t}{T}}. \tag{4.7}$$

The exponential increase variation function begins at 0 and grows to 1 over the duration T. In this function, the number 3 is chosen to provide suitable and clear growth. Conversely, the exponential decrease variation function starts at 1 and decays towards approximately 0.05 (since $e^{-3} \approx 0.05$) over the same interval. Examples of the exponential increase and decrease variation functions applied to sine waves are illustrated in Figure 4.4.



(a) Sine wave with the exponential increase function (4.6) applied, with $a = 0.8$, $f = 0.15$, and $v = 0.2$.



(b) Sine wave with the exponential decrease function (4.7) applied, with $a = 1.0$, $f = 0.2$, and $v = 0$.

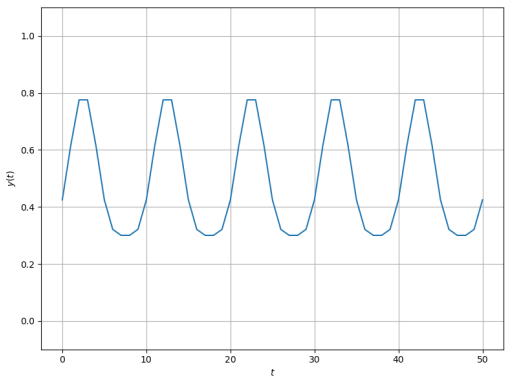
Figure 4.4: Two time series samples of sine waves with exponential increase and decrease amplitude variation functions applied. Both time series are of length 50.

In addition to linear and exponential increases and decreases, several other functions were utilized to simulate diverse variation patterns of the base waves. The following sinusoidal and cosine functions simulate rhythmic variation in the oscillation amplitudes:

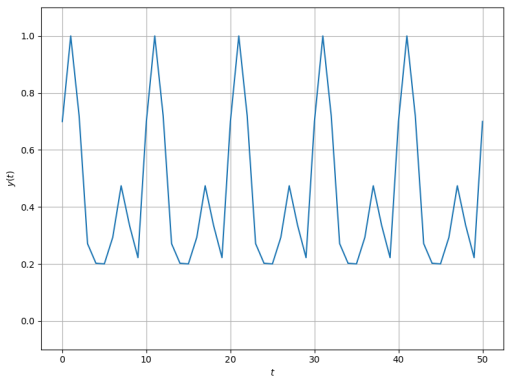
$$A_{\text{sinusoidal}}(t) = 0.5 + 0.5\sin(2\pi \cdot 0.1t), \tag{4.8}$$

$$A_{\text{cosine}}(t) = 0.5 + 0.5\cos(2\pi \cdot 0.1t). \tag{4.9}$$

Figure 4.5 shows examples of using these sinusoidal and cosine amplitude variation functions on a sine wave.



(a) Sine wave with the sinusoidal amplitude variation function (4.8) applied, with $a = 0.5$, $f = 0.1$, and $v = 0.3$.



(b) Sine wave with the cosine amplitude variation function (4.9) applied, with $a = 1.0$, $f = 0.2$, and $v = 0.2$.

Figure 4.5: Two time series samples of sine waves with sinusoidal and cosine amplitude variation functions applied. Both time series are of length 50.

Furthermore, randomness is introduced by a function that creates a smooth, varying random amplitude variation. First, a noise sequence of uniformly distributed random values between 0 and 1 is generated for each time point t:

$$n(t) \sim \text{Uniform}(0,1).$$

These values are smoothed using a moving average with a window of length 5:

$$m(t) = n(t) \cdot \frac{1}{5}[1,1,1,1,1].$$

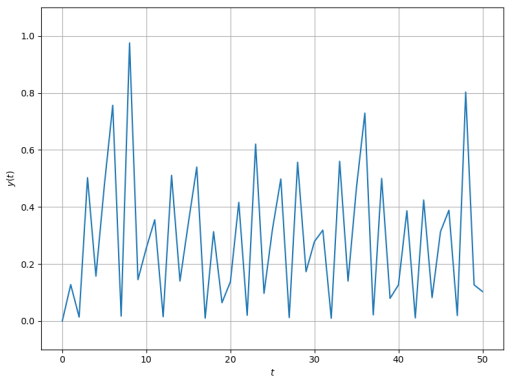
Finally, the smoothed values are normalized to the interval $[0,1]$:

$$A_{\text{random}}(t) = \frac{m(t) - \min(m(t))}{\max(m(t)) - \min(m(t))}. \tag{4.10}$$

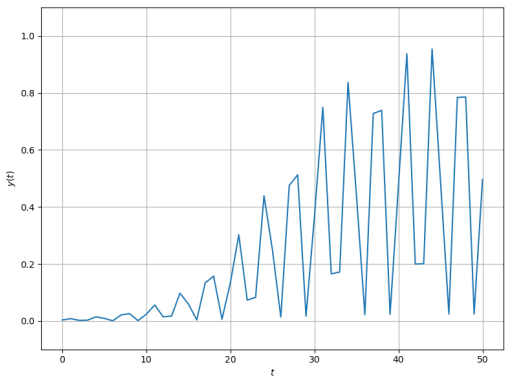
Figure 4.6a shows an example of using the random amplitude variation function. Moreover, a logistic amplitude variation function simulates gradual activation, characterized by a sigmoid curve. The logistic function is defined as:

$$A_{\text{logistic}}(t) = \frac{1}{1 + \exp\left(-10\left(\frac{t}{T} - 0.5\right)\right)}. \tag{4.11}$$

This modulation smoothly transitions amplitude values from near 0 to near 1 around the midpoint $t = T/2$, modeling scenarios involving gradual increases. An example of integrating this function is illustrated in Figure 4.6b.



(a) Sine wave with the random amplitude variation function (4.10) applied, with $a = 1.0$, $f = 0.4$ and $v = 0$.



(b) Sine wave with the logistic amplitude variation function (4.11) applied, with $a = 1.0$, $f = 0.3$ and $v = 0$.

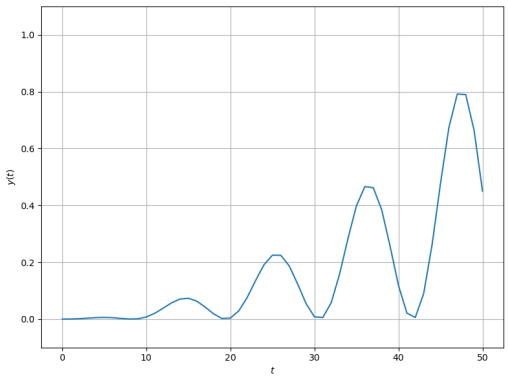
Figure 4.6: Two time series samples of sine waves with random and logistic amplitude variation functions applied. Both time series are of length 50.

The final amplitude variation functions are one quadratic function and one square root function. They simulate nonlinear amplitude variations that represent different rates of signal growth. They are defined as:

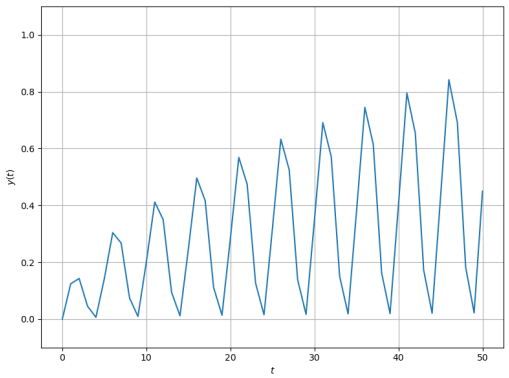
$$A_{\text{quadratic}}(t) = \left(\frac{t}{T}\right)^2, \tag{4.12}$$

$$A_{\text{square\_root}}(t) = \sqrt{\frac{t}{T}}. \tag{4.13}$$

The quadratic function produces a gradual initial growth in amplitude that accelerates towards the end of the interval. The square root function yields rapid initial amplitude growth that gradually slows as the signal approaches the end of the time interval. Samples of integrating these functions are shown in Figure 4.7.



(a) Sine wave with the quadratic amplitude variation function (4.12) applied, with $a = 0.9$, $f = 0.09$ and $v = 0$.



(b) Sine wave with the square root amplitude variation function (4.13) applied, with $a = 1.0$, $f = 0.3$ and $v = 0$.

Figure 4.7: Two time series samples of sine waves with quadratic and square root amplitude variation functions applied. Both time series are of length 50.

Trend Functions

Upward and downward trend shifts are applied to simulate long-term drifts in the oscillatory time series. These functions gradually shift the overall level of the signal over time and are added after amplitude variations. The final waveform is computed as:

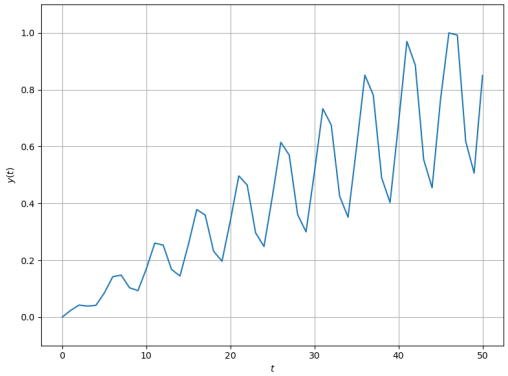
$$y_{\text{final}}(t) = y_{\text{modified}}(t) + \tau(t). \tag{4.14}$$

The upward and downward trend functions are defined as:

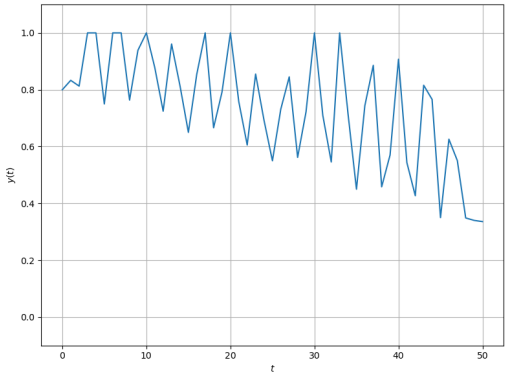
$$\tau_{\text{up}}(t) = \frac{0.5t}{T}, \tag{4.15}$$

$$\tau_{\text{down}}(t) = -\frac{0.5t}{T}. \tag{4.16}$$

These linear trend components shift the signal upward or downward throughout the duration T. Finally, to remain in the interval $[0, 1]$, clipping was performed if the function exceeded that boundary. Figure 4.8 shows two examples with applied trend shifts.



(a) Sine wave with the linear increase amplitude variation function (4.4) and the upward trend function (4.15), with $a = 0.9$, $f = 0.09$, and $v = 0$.



(b) Cosine wave with the random amplitude variation function (4.10) and the downward trend function (4.16), with $a = 0.7$, $f = 0.3$, and $v = 0.8$.

Figure 4.8: Two time series samples of sine and cosine waves with applied amplitude variation and trend functions. Both time series are of length 50.

Table 4.2 provides an overview of the different configurations applied to create the synthetically oscillating time series.

Table 4.2: Configuration table for synthetic time series generation.

Parameter	**Values**
Wave types	sine, cosine
Frequencies	[0.08, 0.4]
Amplitudes	[0.3, 1.0]
Vertical offsets	[0.0, 0.7]
Time series lengths	[40, 140]
Amplitude variation functions	none, linear increase, linear decrease, exponential, increase, exponential decrease, sinusoidal, cosine, random, logistic, quadratic, square root
Trend functions	none, upward, downward

4.2.3 Test Sets

To evaluate the performance of the oscillation detection methods, a custom test set was created by augmenting actuator data, from field test vehicles, through the insertion of both manually discovered oscillations, described in Section 3.4, and synthetically generated oscillations, explained in Section 4.2.2. The original time series, in which the oscillations were inserted, is data from one week from four different vehicles, including approximately 12,000,000 data points in total. Table 4.3 shows the number of data points for each vehicle.

Table 4.3: Number of data points per vehicle time series.

Vehicle	Number of data points
1	3,295,000
2	1,013,000
3	6,274,000
4	1,390,000

The augmentation involved inserting oscillatory segments randomly into the original actuator time series data, replacing the existing segments of the time series at the insertion points. A boundary similarity criterion was applied to ensure realistic transitions between the original and inserted segments, requiring the inserted oscillation boundaries to closely match the corresponding points in the original time series with a predefined tolerance set to 0.1. This process prevented abrupt changes at segment boundaries.

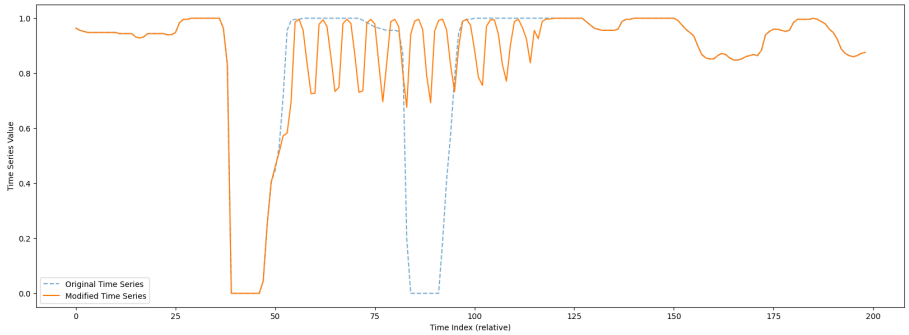
The processes of inserting the manually discovered oscillations and the synthetic oscillations differed slightly. Due to the limited number of available manually discovered oscillatory segments, these were repeatedly reused in the augmentation process. To ensure balanced representation, each segment was sequentially selected and attempts were made to identify suitable insertion points within the original time series. In contrast, given the larger availability of synthetically generated oscillations, each synthetic segment was inserted only once per batch of data. For the OC-SVM dataset, each batch consisted of approximately 1,000,000 data points. For the TranAD dataset, a batch corresponded to one full vehicle time series, and synthetic segments were inserted directly into each of these without further subdivision. In this insertion procedure, a random segment from the original time series was selected first, and then a suitable synthetic oscillation was sought from the available synthetic dataset that would fit smoothly at that location.

Moreover, it was important for the synthetically generated data to include diverse oscillation types. Since some synthetic oscillations were easier to insert smoothly than others, particularly those without added trend shifts compared to those with trend shifts, they were categorized into distinct groups, differentiating between time series with and without trend shifts and between segments with or without an amplitude variation function. Each of these categories was assigned a specific proportion for insertion into the test set, ensuring varied coverage.

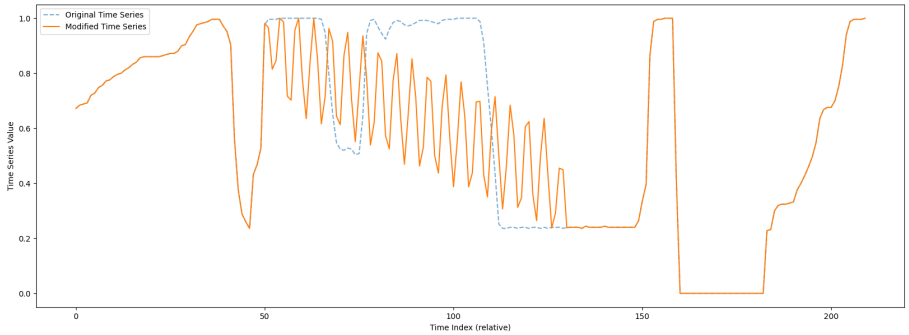
To create labels for the test set, we used a rule based on the proportion of oscillatory points within each window. Specifically, a window was labeled as anomalous if more than 50% of its time points fell within an oscillatory segment. If this threshold was not met, the window was labeled as normal. Two variants of the test set were created, each containing approximately 1% oscillatory content:

- A dataset consisting of 1% of manually discovered oscillations.
- A balanced dataset with 0.5% each of manually discovered real oscillations and synthetically generated oscillations.

Figure 4.9 shows two examples of time series segments, including inserted oscillations.



(a) Manually discovered oscillation inserted in the actuator time series.



(b) Synthetic oscillation inserted in the actuator time series.

Figure 4.9: Two samples of segments of the original time series and corresponding modified time series with inserted oscillations, used as test set. The blue dashed lines correspond to the original time series, while the orange lines correspond to the modified time series.

4.3 Model Implementations

This section describes the implementation of the two anomaly detection models used in this thesis: OC-SVM and TranAD. Since the available data is unlabeled, both models are applied in an unsupervised setting, where the goal is to identify deviations

from normal patterns without access to ground truth annotations. Sections 4.3.1 and 4.3.2 detail the implementation process for each model, respectively, including data preparation, architecture, and hyperparameter settings.

4.3.1 One-Class Support Vector Machine

The time series data were transformed into a set of overlapping vectors using the sliding window approach described in Section 2.4.2. Each vector represented a segment of the time series and served as a single input sample for the model. The window size was set to 30 time steps based on discussions with domain experts, although it would ideally be set dynamically, since oscillations can differ in both duration and frequency. Moreover, the windows were generated with a stride of 1 to ensure high resolution. OC-SVM does not consider temporal dependencies between consecutive windows, meaning the order of the windows is irrelevant. Therefore, data from all vehicles could be combined during training.

OC-SVM was implemented using scikit-learn [45], which provides flexibility in configuring various hyperparameters. Among these, the choice of kernel is particularly important, with the most commonly used options being the RBF and Polynomial kernels. When a polynomial kernel is selected, an additional hyperparameter, the degree of the polynomial, must also be specified.

Beyond the choice of kernel and polynomial degree, different values of ν and γ were investigated. To efficiently tune these hyperparameters, Optuna was employed with a custom objective function. This function was based on the decision scores produced by the model's decision function when applied to the training data. Specifically, the objective function was defined as

$$\text{score} = \text{Var}(f(X_{\text{train}})) - \mu(f(X_{\text{train}})), \tag{4.17}$$

where

- $f(X_{\text{train}})$ represents the decision scores of the model on the training data,
- $\text{Var}(\cdot)$ denotes the variance of these scores,
- $\mu(\cdot)$ represents the mean of the scores.

The decision function of OC-SVM represents the signed distance between a data point and the decision boundary. By convention, inliers have positive decision scores, while outliers have negative scores.

With Optuna, the goal was to minimize the objective function. This corresponds to minimizing the variance of the decision function values while encouraging a higher mean. Minimizing variance ensures that inliers are more compactly clustered together, leading to a more stable decision boundary. Meanwhile, a higher mean encourages a greater proportion of inliers, reducing the risk of overly conservative anomaly detection.

A key challenge in using the decision function as part of the objective function is that the distances it produces are highly dependent on the choice of kernel. Since the RBF and Polynomial kernels map data into different feature spaces, the decision function values are not directly comparable across these kernels. As a result, models trained with different kernels may yield significantly different score distributions, making a direct comparison difficult under a single objective function.

To address this issue, models using the RBF and polynomial kernels were tuned and evaluated separately. This ensured that the hyperparameter tuning process remained consistent within each kernel type, without unfairly favoring one over the other due to differences in decision function scaling. The choice of the best performing kernel was only determined at a later evaluation stage, when the models were tested under real conditions.

The hyperparameter search space is defined in Table 4.4. Hyperparameter tuning is a time-consuming process, and given that the primary goal of this thesis is to evaluate the feasibility of the models rather than to find the optimal configuration, the search space was limited to a manageable range to avoid excessively long tuning times.

When using Optuna, the maximum number of trials can be restricted to limit computational cost. In this thesis, the number of trials was restricted to 5 per tuning run, which is a relatively low number. A higher trial count would likely be necessary if the goal were to find an optimal model rather than assess feasibility. The tuning process was repeated multiple times, and not all polynomial degrees were included in every run. Early experiments showed that degrees 2 and 5 yielded suboptimal results, leading to a focus on degrees 3 and 4 in later trials.

The hyperparameter ν is related to the expected fraction of outliers in the dataset, as described in Section 2.5.2. Since the number of oscillations in the data was unknown, an initial estimate was made together with domain experts at Volvo Group. This helped narrow down the range of ν values.

At early stages of the tuning process, γ was set to either 'scale' or 'auto', which are defined as:

$$\gamma_{scale} = \frac{1}{n_{features} \cdot \mathrm{Var}(X)}, \tag{4.18}$$

$$\gamma_{auto} = \frac{1}{n_{features}}. \tag{4.19}$$

However, both 'scale' and 'auto' resulted in very small γ values, leading to a model that was too constrained and unable to capture the complexity of the data. Therefore, fixed values of γ in the range [0.01, 1] were tested instead.

Table 4.4: Hyperparameter search space used in Optuna tuning.

Hyperparameter	Tested Values
Kernel	RBF, Polynomial
Degree (d)	$\{2, 3, 4, 5\}$
ν	$[0.01, 0.1]$
γ	$[0.01, 1]$

The final hyperparameter configuration for the model with the RBF kernel can be seen in Table 4.5, along with the corresponding objective function score. For the model with the polynomial kernel, the final configuration can be seen in Table 4.6.

Table 4.5: Final hyperparameter configuration for OC-SVM with the RBF kernel.

Hyperparameter	Final Value
Kernel	RBF
ν	0.0178
γ	0.8763

Table 4.6: Final hyperparameter configuration for OC-SVM with the Polynomial kernel.

Hyperparameter	Final Value
Kernel	Polynomial
Degree (d)	3
ν	0.0142
γ	0.0300

4.3.2 TranAD

A modified version of TranAD was implemented with the GitHub repository provided by the authors as a reference [68]. Their implementation was developed using PyTorch [69], which was also used in this thesis to maintain consistency with the original implementation, while also enabling GPU usage. The repository included all essential components of the model, such as data preprocessing, the TranAD architecture, the anomaly scoring, and the POT evaluation. Several adaptations were made to tailor the implementation to the specific task of this thesis.

Preprocessing

The input to TranAD is one input window and one context window, where the input window is part of the context window. The context window is larger and enables the model to more accurately reconstruct the input window. Effectively, this means that the data had to be converted to a dataset of context windows, from which the input windows could be extracted at a later stage. The size of the context windows was set to 300, which corresponds to 30 seconds of data, and the size of the input windows was set to 30, the same as for OC-SVM.

Model Training

The model trained on field test data was trained sequentially on data from multiple trucks, meaning it was first trained on data from one truck before proceeding to the next. Given the large data set, this approach, although unconventional in deep learning, offered a simple way to manage memory constraints. A drawback of this method is that data encountered later in the process has a bigger influence on the final model, compared to data encountered at an earlier stage. However, for the purposes of this study, this trade-off was considered acceptable.

When training on field test data, the data was divided into a training set and a validation set using a 90/10 split for each truck. Early stopping was used to prevent overfitting and reduce training time. Specifically, training would proceed to the next truck if there was no improvement in validation loss for three consecutive epochs. Training was set to run for 15 epochs per vehicle, but early stopping was usually triggered before that.

In contrast, when training on the test cell data memory constraints were not a concern due to the smaller dataset size. As a result, the model could be trained in an offline process rather than sequentially. Other training parameters, including early stopping and epoch limits, remained the same.

Training loss was computed using the reconstruction from phase 2, as defined in equation 2.21, with ϵ set to 1. In the TranAD paper [23], anomaly scores for unseen windows are defined as a combination of the reconstruction losses from phase 1 and phase 2. However, in practice, the anomaly scores were implemented as the squared L2 norm:

$$s_{W_t} = (\hat{O}_2 - W_t)^2. \tag{4.20}$$

Here, $\hat{O}_2$ is the output from phase 2 and W_t is the input window. This simplified implementation was used in this thesis as well. Using the output from both phases might yield a more robust anomaly score, but the squared L2 norm was deemed sufficient for this thesis, while also remaining consistent with the original implementation.

Hyperparameters

The implementation of TranAD involves several hyperparameters that affect both performance and training stability. As in the original implementation, the AdamW optimizer, which is a version of the Adam optimizer with decoupled weight decay [70], was used. A step-based learning rate scheduler was used to reduce the learning rate at fixed intervals to improve training stability. This setup is consistent with the original implementation, but hyperparameter values were tuned for this application. Table 4.7 outlines the configuration used in this thesis.

A subset of hyperparameters was set using Optuna, with the objective of minimizing the reconstruction loss, as defined in equation 2.22, on a validation split of the data.

Table 4.7: Training configuration for TranAD

Hyperparameter	Value
Optimizer	AdamW
Learning Rate	0.0021
Optimizer Weight Decay	0.0016
Step-scheduler Step Size	4
Step-scheduler Gamma	0.5952
Dim. of Feedforward Layers	64
Attention Heads	Number of features
Batch Size	256

The number of trials was limited to 5. Furthermore, the optimization was performed on the test cell data, which was selected due to its smaller size and thus reducing the computational cost. An alternative approach to better align with the target domain could have been to use a subset of the field test data for the optimization. However, for the purpose of this thesis, using the test cell data was deemed sufficient as the primary goal was not to find the optimal configuration, but to investigate the general suitability of the model. The search space used in the Optuna optimization can be seen in Table 4.8. The range for the learning rate was kept narrow, as lower initial learning rates were found to slow down convergence. Similarly, the gamma range was initially set to [0.1, 0.9], but was later set to [0.4, 0.9] since lower gamma values significantly slowed down convergence.

Table 4.8: Hyperparameter search space used in Optuna optimization

Hyperparameter	Range/Values
Learning Rate	$1 \cdot 10^{-2}$ – 1×10^{-3}
Step-scheduler Gamma	1×10^{-2} – 1×10^{-6}
Step-scheduler Step Size	1 – 5
Gamma	0.1 – 0.9
Dim. of Feedforward Layers	{16, 32, 64}

Some architectural and training-related parameters remained unchanged from the original TranAD implementation. Notably, the number of attention heads in the multi-head attention block was set to equal the dimension of the input and dropout was fixed at 0.1.

4.4 Model Evaluation

The evaluation of the models is based on their ability to detect oscillatory behavior in time series data by classifying windows as either inliers or outliers. Each time series in the test set corresponds to data from a single vehicle and is evaluated separately. Each time series contains between 1,000,000 and 6,000,000 data points as described in Section 4.2.3, and since a sliding window with stride 1 is used, nearly as many overlapping windows are generated.

Each window W_t is scored independently with an anomaly score s_{W_t}. Although each window covers a local range of points, evaluating window-level predictions independently does not fully capture the structure of collective anomalies, which typically span extended contiguous intervals. To address this, the evaluation proceeds in two stages:

1. Window-based evaluation, where metrics are computed directly from the set of binary window predictions $\hat{y}_t$ and window scores s_{W_t}.

2. Point-based and segment-based evaluation, where predictions are aggregated back to point-level predictions, enabling range-based metrics and segment-based analysis.

4.4.1 Window-Based

At window-level, the model outputs an anomaly score s_{W_t} for each window. For OC-SVM, the binary predictions $\hat{y}_{W_t} = \{0, 1\}$ are generated by thresholding the decision function output, where a window is classified as anomalous if its score $s_{W_t} < 0$. For TranAD, binary window predictions are obtained using the POT method, which fits a GPD to the training anomaly scores. The threshold r was set to the 85th percentile of the training scores ($p = 0.85$), and q was set to 10^{-5}, in line with the original TranAD implementation.

Precision, recall, F1 score, and AUC are used to measure performance at the window-level. In this case, precision measures the proportion of windows predicted as anomalous that are labeled as anomalous. Recall measures the proportion of all labeled anomalies that are correctly predicted. The F1 score combines these two, and AUC provides an overall score of the model's ability to distinguish between normal and anomalous windows based on the anomaly scores s_{W_t}. For OC-SVM, the AUC calculation is based on the method used in [71], using the model's decision function. Since lower decision scores indicate more anomalous points, the scores are inverted so that higher values correspond to a greater probability of being anomalous. Similarly, for TranAD, the model's final anomaly score is used for AUC calculation.

Accuracy is not reported, as it is not informative in this context. Since approximately 99% of the data is non-anomalous and only about 1% is labeled as anomalous, a high accuracy could easily be achieved by predicting everything as normal.

4.4.2 Point-Based and Range-Based

To better capture collective anomalies, the second evaluation step aggregates window-level predictions to the point level. Each point x_t is associated with multiple overlapping windows. Its point-level score s_{x_t} is computed by averaging binary window predictions that cover it:

$$s_{x_t} = \frac{1}{|\mathcal{W}_t|} \sum_{t' \in \mathcal{W}_t} s_{W_{t'}}, \tag{4.21}$$

where $\mathcal{W}_t$ is the set of windows that include point x_t. For OC-SVM, a point is predicted as anomalous if $s_{x_t} < 0$. For TranAD, POT is used to derive the binary predictions $\hat{y}_{x_t} = \{0, 1\}$, as for the window-level predictions. Based on the point-level predictions, range-based precision and range-based recall are computed. The weighting parameters were set to $\alpha = 0.2$ and $\beta = 0.8$, since "existence" was considered less important than "size" and "cardinality" in the context of oscillatory anomalies.

4.4.3 Segmentation and IoU Matching

To further evaluate the ability to detect oscillatory events, consecutive predicted anomalous points are grouped into anomaly segments, which are then matched against ground truth anomaly segments using IoU matching. A segment is allowed to consist of only one single anomalous point.

For each ground truth segment R, all predicted segments $\mathcal{P}$ that partially overlap it are identified. The IoU is calculated as the total length of the overlapping regions between these predicted segments and the ground truth segment, divided by the length of their combined union:

$$\text{IoU}(\mathcal{P}, R) = \frac{\sum_{P \in \mathcal{P}} |P \cap R|}{\left| \left(\bigcup_{P \in \mathcal{P}} P \right) \cup R \right|}. \tag{4.22}$$

A ground truth segment is considered a true positive if the combined IoU with overlapping predicted segments is at least 30%. Ground truth segments that are not sufficiently covered by any predictions are counted as false negatives, and predicted segments that do not contribute to any matched ground truth segment are counted as false positives. This segment-level evaluation provides a more informative measure of performance by focusing on the accurate detection of entire oscillatory periods, rather than isolated or scattered anomalous windows.

5

Results

This chapter reports the experimental results obtained using OC-SVM and TranAD. The results are organized into two sections: Section 5.1 presents the outcomes of the OC-SVM approach, while Section 5.2 provides the performance of the TranAD method.

5.1 One-Class Support Vector Machine

The final OC-SVM models after hyperparameter tuning were tested against each other: one with the RBF kernel and one with the polynomial kernel, as described in Section 4.3.1. The model using the RBF kernel outperformed the polynomial variant and was therefore chosen for all further evaluation. The results presented in this section are based on the OC-SVM model with the RBF kernel, trained on field test data.

Table 5.1 shows the window-level performance of the model across the two dataset types. Overall, OC-SVM performs better on the dataset type that includes synthetic oscillations. All metrics are consistently higher across all vehicles in this group. The actuator data from Vehicle 2 achieves the best overall performance, with a precision of 0.0202, F1-score of 0.0385, and AUC of 0.6149. However, the overall results are poor, where precision, recall, and F1-scores are low across all datasets. Low recall indicates that the model struggles to identify the labeled oscillatory patterns.

Table 5.1: Window-level performance of OC-SVM over the two test set variants across the four vehicle time series. Best results are shown in bold.

Test Set Variants	Vehicle	Precision	Recall	F1-score	AUC
1% manually discovered oscillations	1	0.0110	0.2144	0.0210	0.4273
	2	0.0120	0.2158	0.0228	0.4550
	3	0.0072	0.2330	0.0139	0.2950
	4	0.0036	0.2199	0.0071	0.2804
0.5% manually discovered, 0.5% synthetic oscillations	1	0.0200	**0.4385**	0.0382	0.6101
	2	**0.0202**	0.4149	**0.0385**	**0.6149**
	3	0.0108	0.4020	0.0211	0.4484
	4	0.0096	0.3957	0.0188	0.4486

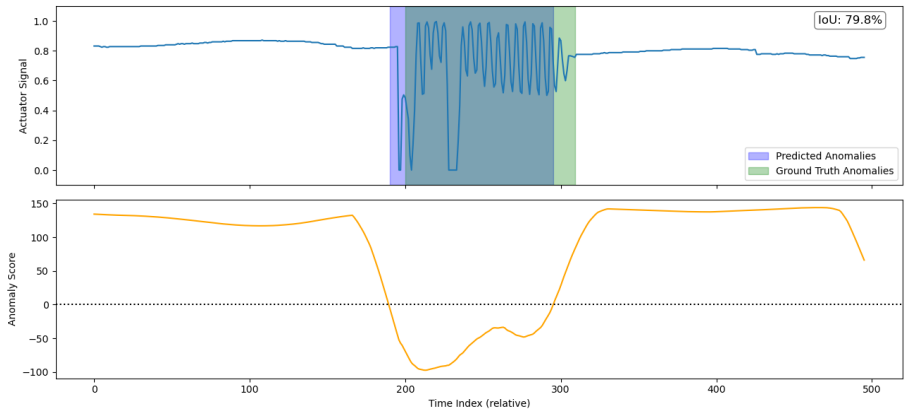
Table 5.2 presents the point-level and segment-wise performance of the OC-SVM model across the two dataset types over the four vehicles. As with the window-level evaluation, the model performs better on the dataset type containing synthetic oscillations. Range-based recall is consistently higher across all vehicles in the mixed dataset, with Vehicle 1 reaching the highest R_X at 0.5242. Similarly, the proportion of true positive segments is higher in the synthetic setting for all vehicles.

Table 5.2: Performance of OC-SVM over the two test set variants across the four vehicle time series. P_X and R_X denote range-based precision and recall, computed from point-level labels. TP, FN, and FP are segment-level counts. Best range-based results are in bold.

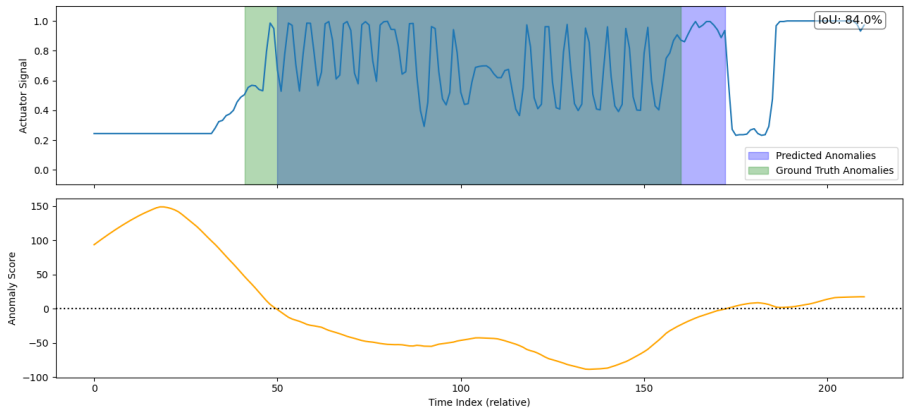
Test Set Variants	Vehicle	P_X	R_X	TP	FN	FP
1% manually discovered oscillations	1	0.0125	0.2598	77	211	9,880
	2	0.0132	0.2550	27	68	3,237
	3	0.0082	0.3044	98	500	23,782
	4	0.0102	0.3086	22	72	3,616
0.5% manually discovered, 0.5% synthetic oscillations	1	0.0194	**0.5242**	177	143	9,883
	2	0.0195	0.4981	57	49	3,250
	3	0.0111	0.4894	213	429	23,694
	4	**0.0240**	0.4885	80	95	3,587

While the range-based precision values are low overall in Table 5.2, mostly below 0.02, this is not necessarily a sign of poor performance. In this work, one of the goals is also to discover potential oscillatory patterns that may not have been labeled, as the datasets are known to be incomplete. From this perspective, a high number of false positive segments could indicate that the model is successfully identifying additional, previously unknown oscillations. Therefore, the low precision values should be interpreted with caution and viewed as a starting point for further inspection rather than a definitive performance limitation.

For further analysis of the results, true positive, false negative, and false positive segments are plotted to visually inspect what the model is able to detect and what it misses. Figure 5.1 illustrates two examples of true positive segment detections by the OC-SVM model. In both these cases, the predicted segments align well with the labeled oscillations, as indicated by the high IoU values. The oscillatory patterns in both cases show clear deviations in the actuator signal, which the model captures. In Figure 5.1a, the anomaly score drops sharply in the region where the oscillation occurs, indicating a strong confidence in the prediction. In contrast, in Figure 5.1b, the anomaly score remains slightly under the threshold throughout the oscillatory segment and reaches its minimum near the peak amplitude of the oscillation. This suggests that the model still captures the anomaly, but with less certainty.



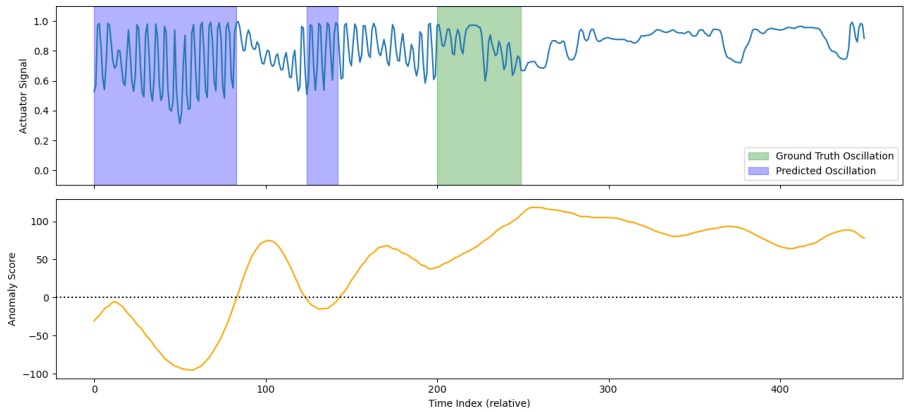
(a) Manually discovered oscillation in data from Vehicle 3.



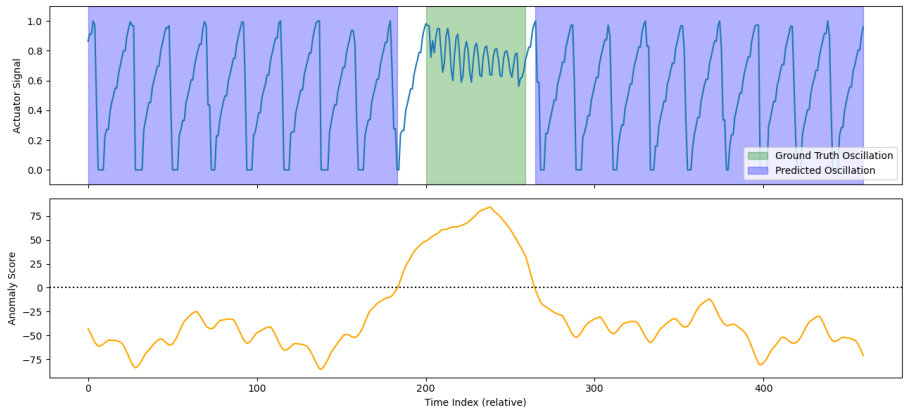
(b) Manually discovered oscillation in data from Vehicle 1.

Figure 5.1: Two examples of true positive detections by OC-SVM based on point-wise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold.

Figure 5.2 illustrates two false negative cases where the model fails to detect the labeled oscillation entirely. In Figure 5.2a, the model fails to detect labeled oscillatory segments with low amplitude, but correctly identifies a higher amplitude, unlabeled oscillation that occurs earlier in the time series. In Figure 5.2b, the model misses the labeled oscillation, but classifies the surrounding as anomalous. However, this particular example comes from Vehicle 3, whose data appears to contain more fluctuations.



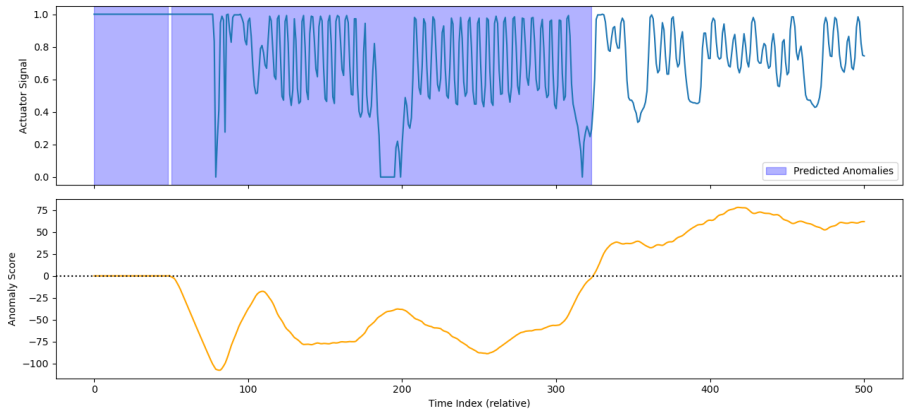
(a) Actuator data from Vehicle 1, with manually discovered oscillation inserted.



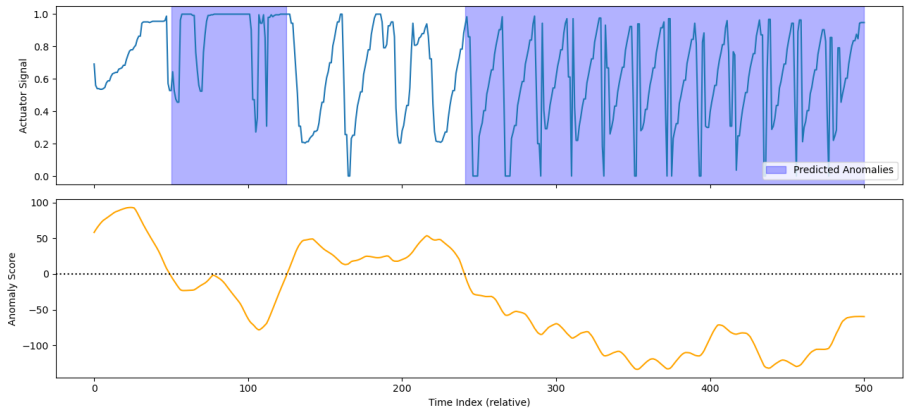
(b) Actuator data from Vehicle 3, with manually discovered oscillation inserted.

Figure 5.2: Two examples of false negative cases from OC-SVM based on point-wise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold.

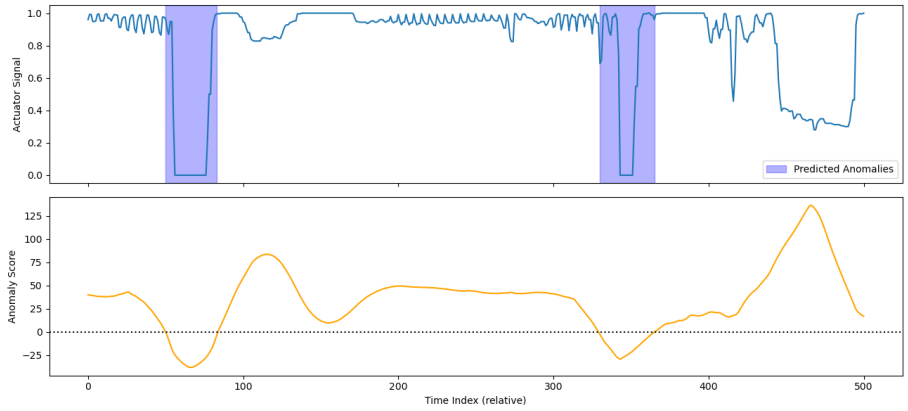
Figure 5.3 shows three false positive cases. In Figure 5.3a, the model successfully detects oscillatory regions, but does not capture the oscillations that follow. In the same figure, there is also a flat segment that the model classifies as anomalous, which is a common type of false positive segment. Figure 5.3b illustrates another example of where the model classifies high amplitudes as anomalies. Finally, Figure 5.3c shows missed oscillatory segments where the amplitudes are very low, but classifies the parts with extreme value changes as anomalous.



(a) Actuator data from Vehicle 1.



(b) Actuator data from Vehicle 3.

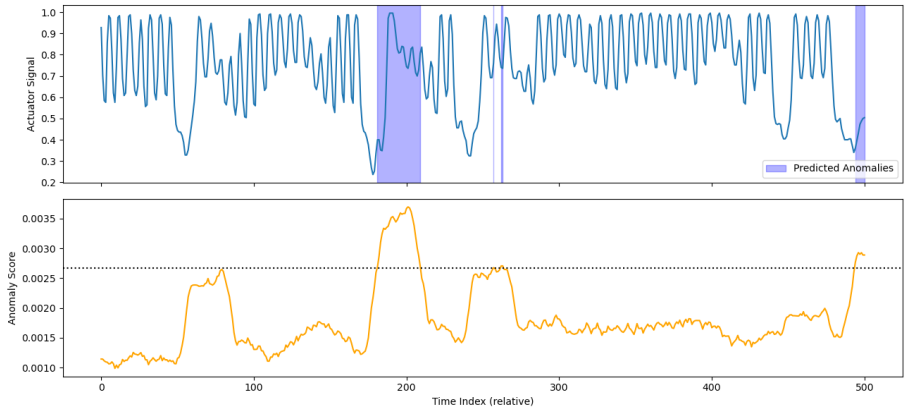


(c) Actuator data from Vehicle 4.

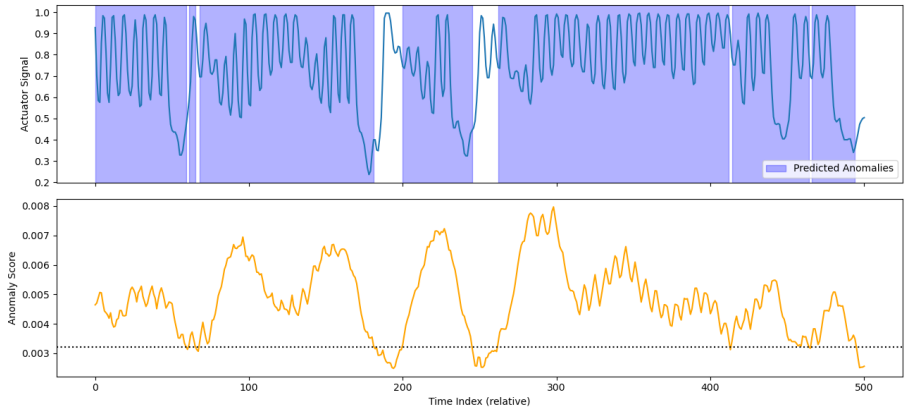
Figure 5.3: Three examples of false positive cases from OC-SVM based on point-wise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold.

5.2 TranAD

Three main TranAD models were developed: two univariate models trained on field test data and test cell data, respectively and one multivariate model trained on test cell data. The training and validation losses of these are shown in Appendix A.1 and A.3.



(a) Predictions and anomaly scores from TranAD trained on field test data.



(b) Predictions and anomaly scores from TranAD trained on test cell data.

Figure 5.4: Comparison of oscillation predictions and anomaly scores from TranAD models trained on field test data and test cell data. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold.

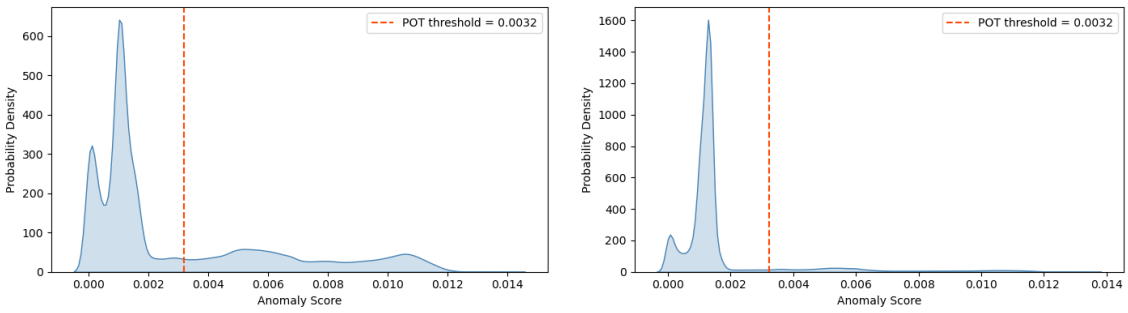
When comparing the performance of the univariate models, it was concluded that the model trained on test cell data was more successful in detecting oscillatory segments. Figure 5.4 illustrates this with two plots of the same actuator signal segment. Figure 5.4a shows predictions and anomaly scores from the model trained on field test data, while Figure 5.4b shows results from the model trained on test cell data. As seen on the anomaly scores, the model trained on field test data tends to

classify abrupt changes as anomalies rather than oscillations. In contrast, the model trained on test cell data is more sensitive to oscillatory patterns. Based on visual observations, the model trained on test cell data was selected for further evaluation. The remaining results in this section are therefore based on this model.

Table 5.3 presents the window-level performance of TranAD across the two test set variants and four vehicle time series. The model shows a consistent pattern of relatively high recall, typically between 0.45 and 0.49, but low precision across all vehicles, resulting in low F1-scores. This indicates that TranAD detects many true anomalies but also produces a large number of false positives.

Table 5.3: Window-level performance of TranAD over the two test set variants across the four vehicle time series. Best results are shown in bold.

Test Set Variants	Vehicle	Precision	Recall	F1-score	AUC
1% manually discovered oscillations	1	0.0164	0.4788	0.0317	0.5553
	2	0.0188	**0.4925**	0.0361	0.5340
	3	0.0151	0.4733	0.0293	0.5461
	4	0.0454	0.4729	0.0828	**0.6338**
0.5% manually discovered, 0.5% synthetic oscillations	1	0.0176	0.4779	0.0340	0.5682
	2	0.0161	0.4811	0.0312	0.5440
	3	0.0160	0.4838	0.0310	0.5604
	4	**0.0640**	0.4528	**0.1121**	0.6251



(a) Vehicle 1. **(b)** Vehicle 4.

Figure 5.5: Density distributions of window-level anomaly scores for Vehicle 1 and Vehicle 4 in the datasets containing 1% manually discovered oscillations. The vertical dashed line indicates the threshold obtained using POT.

Among the vehicles, Vehicle 4 stands out with noticeably higher precision and F1-scores, especially in the mixed dataset scenario, where it reaches a precision of 0.0640 and an F1 score of 0.1121. A likely reason is that the model detects fewer unlabeled anomalous segments for this vehicle, which reduces the number of false positives. This is reflected in the distribution of window-level anomaly scores shown in Figure 5.5, where Vehicle 4 displays a more concentrated distribution with less mass in the high-score region compared to Vehicle 1. The same pattern is observed in

the range-based evaluation, in Table 5.4, where Vehicle 4 again achieves the highest range-based precision, despite having recall values similar to the other vehicles. Additionally, Vehicle 4 yields the highest AUC scores across both test set variants, suggesting that TranAD is more effective at distinguishing anomalous windows in this particular signal.

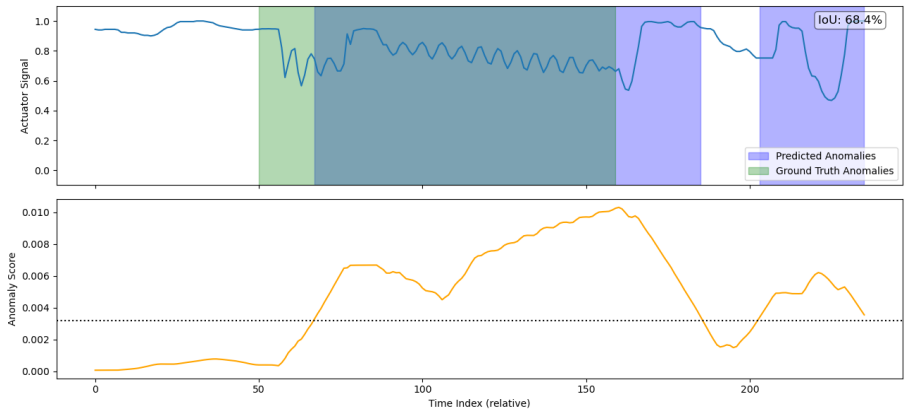
Table 5.4: Performance of TranAD on the two test set variants across the four vehicle time series. P_X and R_X denote range-based precision and recall, computed from point-level labels. TP, FN, and FP are segment-level counts. The best range-based results are shown in bold.

Test Set Variants	Vehicle	P_X	R_X	TP	FN	FP
1% manually discovered oscillations	1	0.0201	0.5280	167	71	25,411
	2	0.0249	0.5526	57	39	7,621
	3	0.0103	0.5162	327	120	98,885
	4	0.0485	0.5299	71	22	4,307
0.5% manually discovered, 0.5% synthetic oscillations	1	0.0244	0.5441	233	83	25,395
	2	0.0228	**0.5622**	75	34	7,576
	3	0.0108	0.5536	427	162	98,838
	4	**0.0749**	0.5172	120	42	4,291

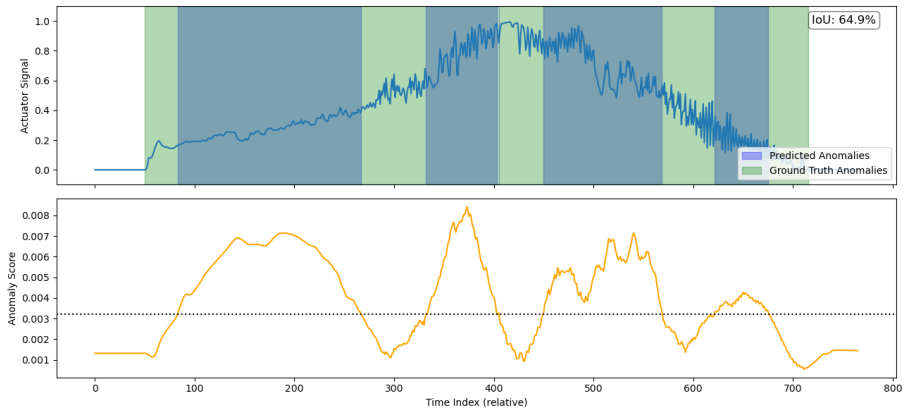
In contrast, Vehicle 3 presents a more challenging scenario for TranAD. The actuator signal for this vehicle contains frequent, irregular fluctuations with significant up-and-down behavior, which do not consistently match the oscillatory patterns seen in the labeled anomalies. As a result, TranAD struggles to distinguish between normal fluctuations and true anomalies, leading to a large number of misclassifications. Even though the dataset for Vehicle 3 is considerably larger than the others, the number of false positives remains very high, with nearly 99,000 incorrect anomaly predictions.

Furthermore, true positive, false negative, and false positive segments are plotted for visual inspection of the model's behavior. Figure 5.6 illustrates three examples of true positive segments detected by TranAD. Figure 5.6a shows an example where TranAD successfully detects a low-amplitude oscillation, while Figure 5.6b shows how the model detects parts of a long labeled oscillation. Figure 5.6c reflects how the model finds a synthetic oscillation with relatively high confidence in the anomaly score.

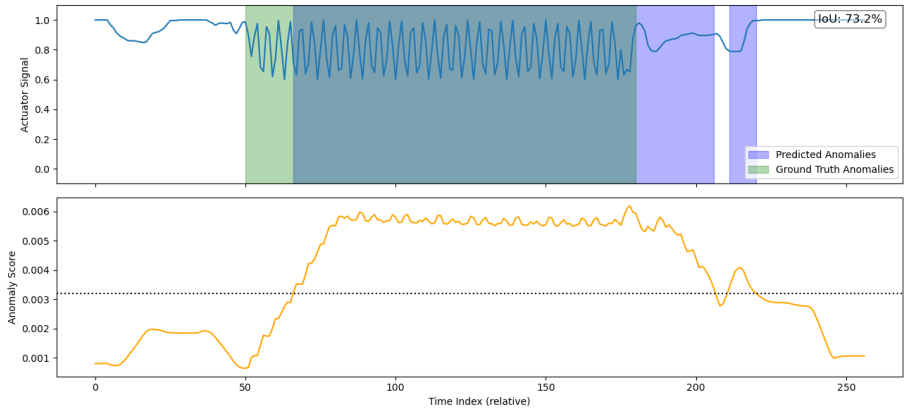
Figure 5.7 shows how the model misses two synthetic labeled oscillations. In Figure 5.7a, the anomaly score drops where the oscillating segment begins, gets higher in the middle of the oscillation, then again lowers. Another observation in that figure is that the oscillating signal has a low value, while most correctly identified oscillations have higher values. In the time series shown in Figure 5.7b, the model first predicts what looks like an unlabed oscillation, then it completely misses the labeled oscillation but predicts the flat signal as anomalous. This ground truth segment also has a relatively low value.



(a) Manually discovered oscillation in data from Vehicle 1.

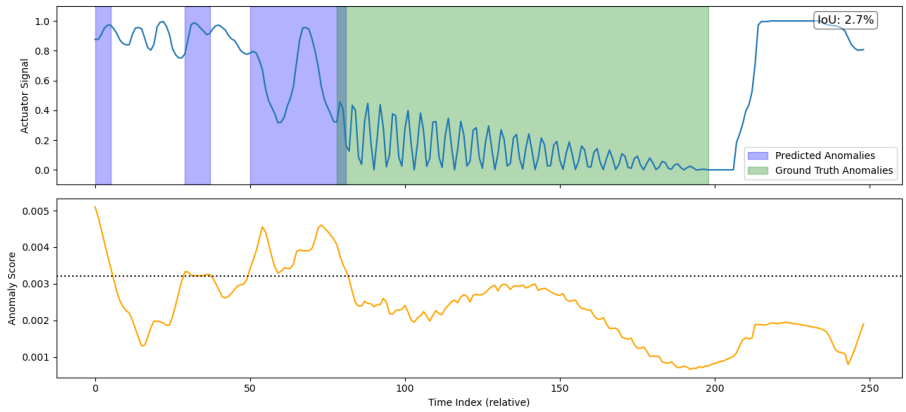


(b) Manually discovered oscillation in data from Vehicle 1.

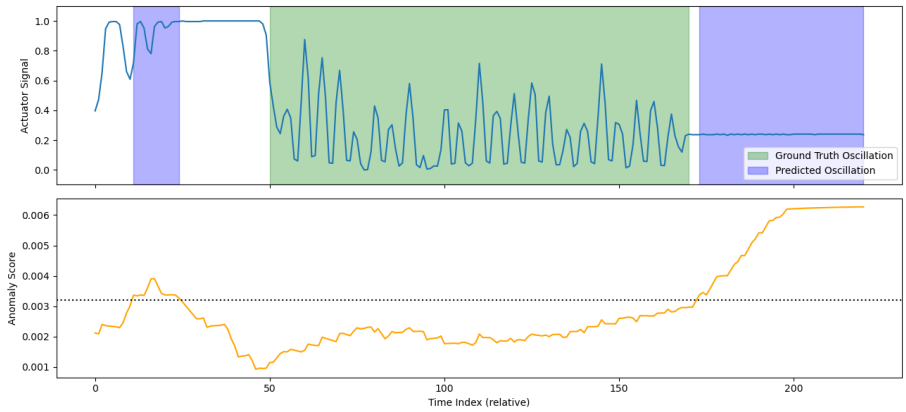


(c) Synthetic oscillation in data from Vehicle 1.

Figure 5.6: Three examples of true positive detections by TranAD based on point-wise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold.



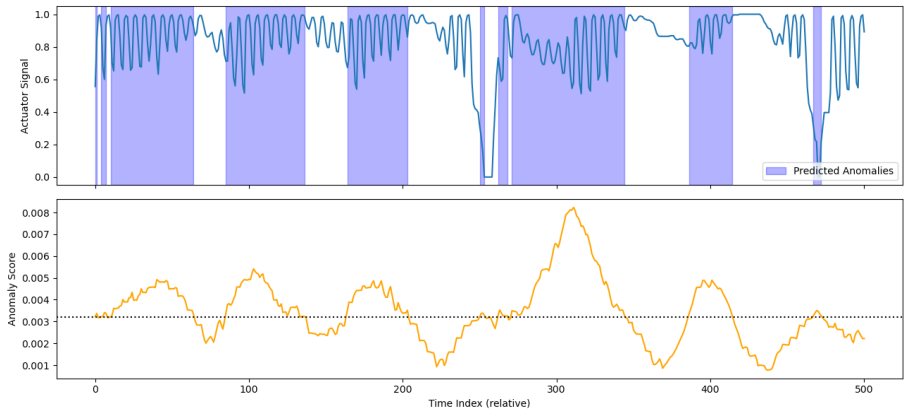
(a) Actuator data from Vehicle 1, with synthetic oscillation inserted.



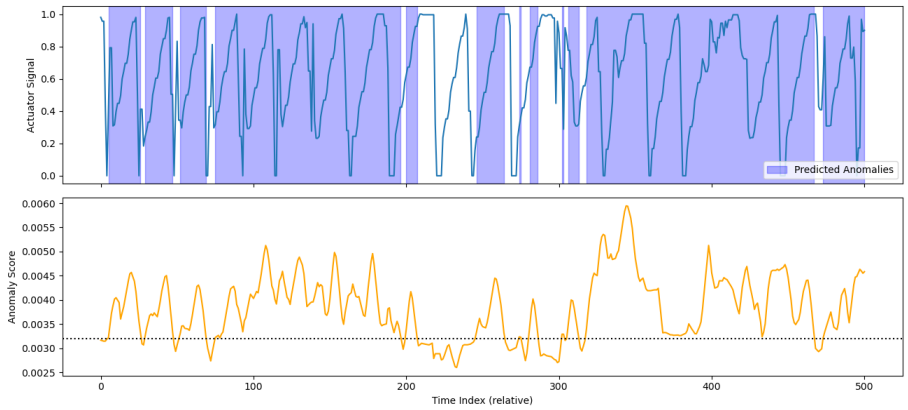
(b) Actuator data from Vehicle 1, with synthetic oscillation inserted.

Figure 5.7: Two examples of false negative cases from TranAD based on point-wise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold.

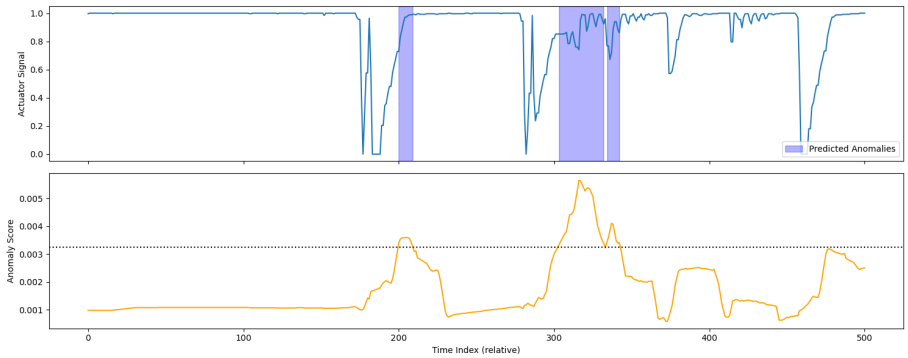
False positive segments are illustrated in Figure 5.8. In Figure 5.8a, the model correctly identified oscillating segments, with clearly higher anomaly scores in the regions where the oscillations appear. However, in Figure 5.8b, the model classifies segments with large fluctuations as anomalous. According to domain experts, these likely originate from another signal. Moreover, Figure 5.8c shows a case where the model finds a short anomalous segment, but misses other short segments of oscillations.



(a) Actuator data from Vehicle 2.



(b) Actuator data from Vehicle 3.

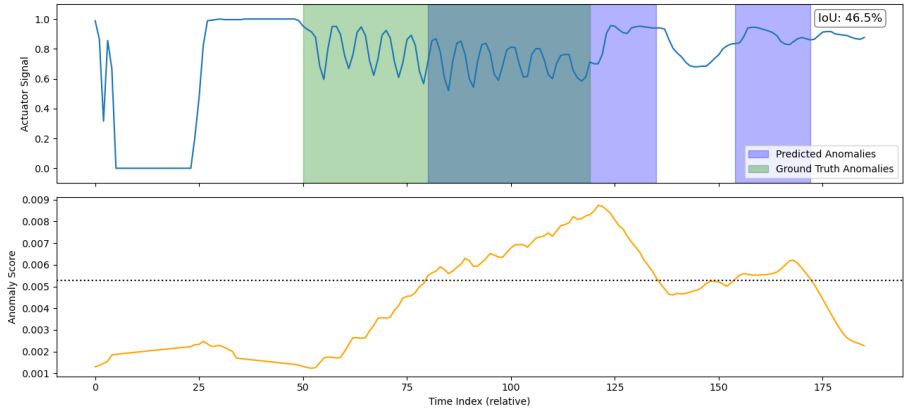


(c) Actuator data from Vehicle 4.

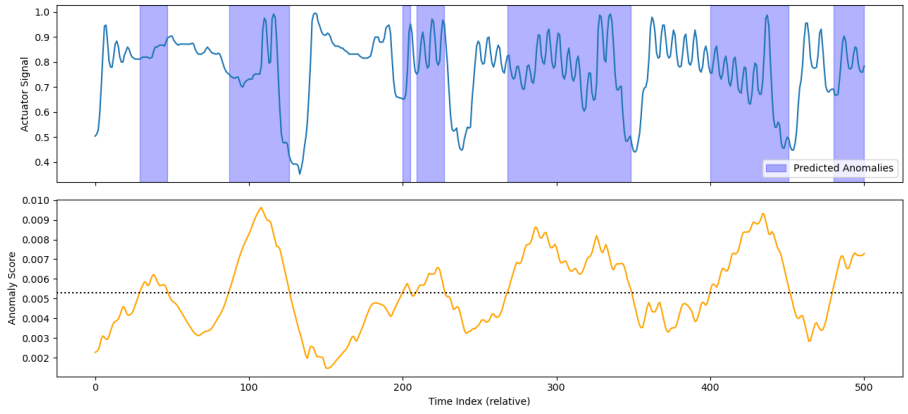
Figure 5.8: Three examples of false positive cases from TranAD based on point-wise predictions. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line in the bottom plot illustrates the threshold.

While the predictions presented so far are obtained using POT with parameters $p = 0.85$ and $q = 10^{-5}$, an additional evaluation is conducted using a stricter threshold to assess the model's robustness. Increasing p to 0.90, while keeping

q constant, results in a higher anomaly score threshold, making the model more selective in what it considers anomalous. As illustrated in Figure 5.9, TranAD still manages to detect oscillatory segments under this stricter setting. However, the detections are less consistent and clear, and some subtle anomalies are missed. Figure 5.9a shows a true positive segment, where the model successfully identifies a synthetic oscillation despite the stricter threshold. In Figure 5.9b, the model predicts an unlabeled oscillatory segment correctly. To support these observations, a quantitative comparison using the stricter POT setting is provided in Appendix B.1 and B.2. As expected, raising the threshold results in lower recall and F1-scores across all vehicles, but precision improves slightly in several cases. It is also noted that the range-based recall is significantly lower than the window-level recall, likely because of a high cardinality factor.



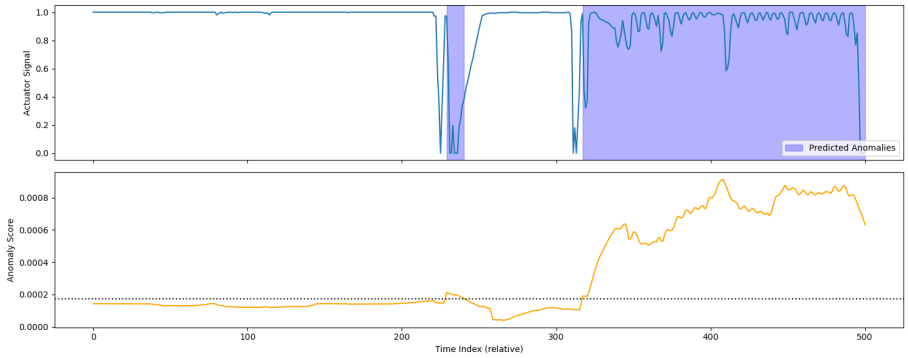
(a) Actuator data from Vehicle 1, with synthetic oscillation inserted, with true positive segment.



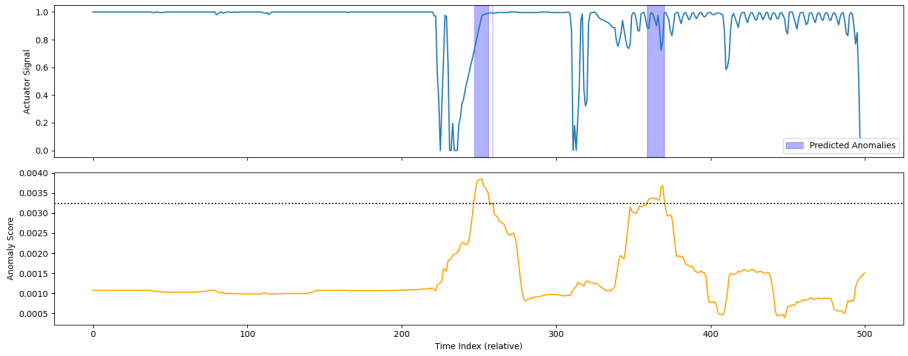
(b) Actuator data from Vehicle 2, with false positive segments.

Figure 5.9: Predictions from TranAD using POT with a stricter threshold, $p = 0.90$, $q = 10^{-5}$. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line represents the threshold.

Finally, a qualitative evaluation of the multivariate TranAD model is done. Only a model trained on test cell data is examined, partly because the test cell trained univariate model is considered the best performing and partly due to time constraints. No quantitative evaluation is performed, as no labeled dataset has been developed for the multivariate case. Instead, the multivariate model is compared visually against the univariate model on the same time intervals. Figure 5.10 presents a case where the multivariate model performs better, capturing a more complete and coherent oscillatory pattern than the univariate version. In contrast, Figure 5.11 shows an example where the univariate model detects oscillating ranges more clearly.

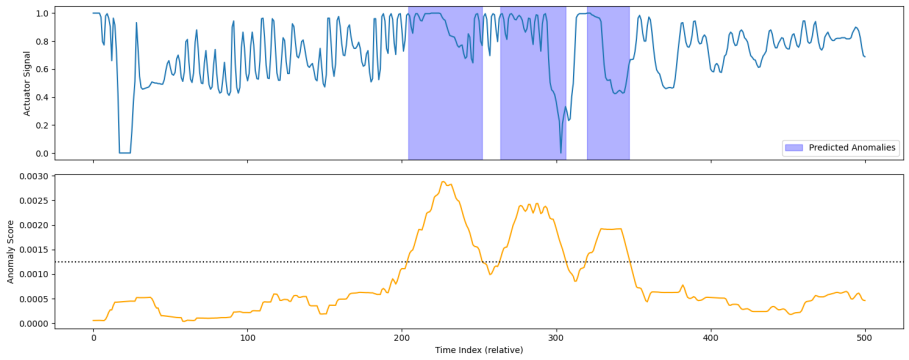


(a) Multivariate.

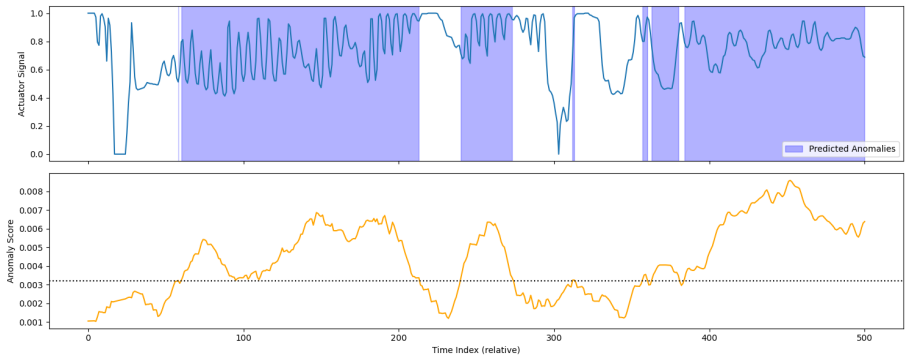


(b) Univariate.

Figure 5.10: Comparison of oscillation predictions and anomaly scores from multivariate and univariate TranAD. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line represents the threshold.



(a) Multivariate.



(b) Univariate.

Figure 5.11: Comparison of oscillation predictions and anomaly scores from multivariate and univariate TranAD. In each figure, the top plot shows the actuator signal, with predicted anomalous segments in blue and ground truth anomalies in green. The bottom plot displays the corresponding anomaly scores s_{x_t} over time. The dashed black line represents the threshold.

6

Discussion and Conclusion

This chapter provides a discussion of the results and methods used in the thesis. Section 6.1 and Section 6.2 discuss the results of the two chosen models. In Section 6.3, limitations and considerations are discussed. Section 6.5 provides potential future directions, and Section 6.6 concludes the thesis.

6.1 One-Class Support Vector Machine

The results from OC-SVM indicate that the model better detects synthetic oscillations than manually discovered oscillations. As shown in Table 5.1 and Table 5.2, this applies to all vehicles. This might indicate that the synthetic segments do not fully capture the complexity of the real-world data. Oscillations detected by OC-SVM, both synthetic and manually discovered ones, are primarily high-amplitude oscillations. This may suggest that the model is more sensitive to extreme values, rather than to the underlying oscillatory patterns. Since OC-SVM treats each window as a single point in n-dimensional space, where n is the window size, it does not consider temporal information. If an oscillation occurs within a typical value range, the corresponding window may not stand out enough in feature space to be classified as anomalous. Contrarily, windows with high-amplitude oscillations with values closer to the extremes are more likely to be classified as anomalous. This highlights a key limitation of OC-SVM, namely that it tends to detect anomalies based on magnitude rather than temporal patterns.

The performance of OC-SVM is highly dependent on the settings of hyperparameters like ν and γ. In this case, ν, which is directly correlated to the fraction of data the model will classify as outliers, was set to 0.0178. Since the actual proportion of oscillations in the data is unknown, it is difficult to assess whether this value is reasonable. Optuna was used to optimize the hyperparameters, but this was a time-consuming process and therefore, the number of trials was limited, making it unlikely that optimal settings were found. Since OC-SVM lacks a conventional loss function, a custom objective function had to be defined. This introduces additional uncertainty as it is difficult to verify whether the defined objective function accurately reflects the model's actual performance. An alternative approach could be to use a labeled validation set for the optimization process, though that introduces new challenges of its own. For instance, the validation set must be representative

of the full dataset to provide reliable guidance during tuning. This requirement was difficult to meet in this thesis due to the lack of labeled oscillations. However, given the results of this work, more oscillations have now been detected and creating a representative validation set may now be more feasible.

These findings highlight both strengths and limitations of OC-SVM in this context. The model is relatively simple, with few hyperparameters, making it easy to implement and interpret compared to more complex approaches. However, the simplicity also limits the model's ability to capture complex behavior, especially subtle patterns such as low-amplitude oscillations. Given its tendency to detect anomalies based on magnitude rather than oscillatory patterns, combined with the high computational cost on large data sets, it is unclear whether significant improvements can be achieved through better hyperparameter tuning. While a better objective function might offer some improvements, the model's inherent limitations suggest that the approach might not be the best suited for this use case.

6.2 TranAD

TranAD demonstrates the capability of detecting oscillating patterns in actuator data. In contrast to OC-SVM, TranAD is able to detect certain low-amplitude oscillations. This is particularly shown in the labeled oscillations, as illustrated with an example in Figure 5.6a, but also in some unlabeled oscillations, as demonstrated with an example in Figure 5.8c. While TranAD is nearly as successful at identifying manually discovered oscillations as it is with synthetic ones, indicating robustness, it still fails to detect some oscillatory patterns and misclassifies normal fluctuations as anomalies.

Across all vehicles, TranAD achieves relatively high recall, typically around 0.45 to 0.56, depending on the test set, indicating that it is able to detect many of the labeled anomalies. However, this comes at the cost of very low precision, often below 0.02, meaning that the model generates a substantial number of false positives. This pattern is reflected in both the window-level and range-based evaluations.

A consistent observation throughout the results is that TranAD tends to predict many short anomalous segments rather than longer contiguous ones. This behavior fragments the detection of anomalies and negatively impacts the range-based recall. In contrast, OC-SVM tends to produce fewer, longer segments that are more likely to overlap completely with the labeled anomalies. This difference becomes particularly relevant under stricter threshold settings, where the range-based evaluation metric penalizes such fragmentation more heavily. As a result, TranAD's range-based recall drops sharply when a higher threshold is applied, as seen in Appendix B.2, using $p = 0.90$ and $q = 10^{-5}$.

Among the evaluated vehicles, Vehicle 4 stands out with significantly better performance. It achieves the highest precision, AUC, and F1-score. The score distribution, provided in Figure 5.5b, shows that the anomaly scores for this vehicle are more concentrated, with less mass in the high-score region. This likely indicates a more stable

signal, where anomalous patterns are more distinct and less likely to be confused with normal fluctuations. In contrast, Vehicle 3 has a more irregular actuator behavior with frequent signal regulation, which likely originates from other signals. This vehicle produces a very large number of false positives, which highlights the model's struggle with generalization across different vehicle types.

The best performing TranAD model was trained on test cell data and showed the most reliable detection performance on Vehicles 1 and 2, whose actuator signals are similar to the controlled environment. However, this strength also reveals a limitation. For signals that differ significantly from the test cell conditions, such as the highly regulated behavior seen in Vehicle 3, the model's predictions become less reliable. This suggests a degree of overfitting to the test cell data, limiting the model's ability to generalize to more complex or diverse real-world data.

Furthermore, it was difficult to determine whether the multivariate model performs better than the univariate one, as no labeled dataset was available for quantitative evaluation, and no suitable metric could be applied. The visual comparisons provide some insight, but are not sufficient for a reliable conclusion. Additionally, the multivariate model in this thesis only incorporated torque and speed signals. Including other relevant signals could potentially improve the model's ability to detect complex oscillatory patterns.

The implementation of TranAD in this thesis followed the original architecture closely, but several design decisions could be reflected. One notable aspect is the sequential training strategy, where the model was trained truck-by-truck. This approach introduces the potential bias that data from later trucks has a greater influence on the final model than data seen earlier. Randomizing the training order or periodically re-training on earlier trucks' data could help create a more balanced model. However, this bias was not evident in the performance of the model trained on field test data and was ultimately not an issue, since the model trained on test cell data was selected for final evaluation.

Another model decision was the choice of context window size, which was 300. Whether this is an appropriate size was not evaluated in this thesis and could be further examined in future work, as it may influence the model's ability to capture relevant temporal patterns.

The hyperparameter tuning process in this thesis involved the use of Optuna to optimize a subset of parameters. While this helped adapt the model to the task at hand, several other architectural and training-related choices were used directly from the original implementation without further investigation. These include the dropout rate, fixed at 0.1, the use of ReLU activations, and the rule of setting the number of attention heads equal to the number of input features. An exploration of these parameters, including different activation functions or attention configurations, might provide more effective setups tailored to this particular task.

Finally, the approach of the anomaly score was simplified in the implementation. Instead of combining the reconstruction errors from both phase 1 and phase 2, as

proposed in the TranAD paper, the score was calculated as the squared L2 norm of the difference between the phase 2 output and the input window. This decision was made for consistency with the authors' codebase and to reduce complexity. While this scoring method was effective for detecting anomalies, using both phases might yield more robust predictions.

TranAD shows strong potential for detecting oscillatory anomalies, successfully capturing many oscillation patterns, including subtle and low-amplitude ones. Its architecture enables advanced pattern recognition, but also results in fragmented predictions. With better tuning, architectural refinements, and training on a more diverse yet controlled dataset, TranAD has the potential to become a more robust and generalizable oscillation detection method.

6.3 Limitations and Considerations

A key limitation with the field test data, used during training, is that it may contain unlabeled oscillations. This affects both OC-SVM and TranAD, as both models rely on learning what normal behavior looks like. OC-SVM assumes that training data consists only of inliers, and TranAD learns to reconstruct and predict normal time series patterns during training. If oscillatory behavior is present during training, the model may learn to treat it as normal, making it harder to detect similar patterns at test time. This issue affects both learning and evaluation and may lead to an underestimation of the models' actual capabilities.

Since the actuator patterns of the different vehicles vary significantly, one possible strategy could be to train separate models based on the specific type of vehicle or driving behavior pattern. This approach might improve performance, as seen with the TranAD model, which was trained on test cell data. As previously discussed, the test cell actuator patterns were more similar to those of certain vehicles, and this model performed better on those vehicles. However, while training models on more similar driving patterns could yield better results for specific cases, it may reduce the generalization of the model. A generalized model that performs well across vehicle types is often preferred. Therefore, although tailoring models to specific driving patterns could enhance performance in controlled settings, it may limit the scalability and flexibility of the model.

Given the scope of this thesis, several design choices were made that likely influenced performance but were not thoroughly explored. For example, window size was fixed at 30 data points in an effort to balance high resolution with the need to capture oscillatory patterns. Similarly, stride was set to 1 to retain as much information as possible and ensure thorough temporal coverage. For smaller datasets, such as the test cell data, this is a reasonable approach to maximize the size of the dataset. However, for larger datasets, such as the field data, this may be less critical and a larger stride could be considered to reduce computational cost without greatly affecting performance.

The training sets used in this thesis vary in size, which might initially appear to

introduce unfairness. However, it is important to note that the primary goal of this work was not to compare the performance of the models directly. Instead, the training sets were curated with a specific model in mind. As such, if a comparative study were to be conducted, the design of the training sets and other experimental settings would require more careful consideration than what was applied in this thesis.

6.4 Evaluation

The evaluation was not completely straightforward, where one main challenge was the unlabeled dataset. Apart from a small number of manually discovered oscillations, the rest of the dataset lacked ground truth labels. To enable evaluation, the manually discovered oscillations were inserted into existing time series data, along with additional synthetically generated oscillatory segments. This made it possible to create a labeled test set, but also introduced limitations. For instance, transitions between normal and inserted segments could result in unnatural edges and disrupt the temporal structure of the data. Additionally, the synthetic oscillations may not fully reflect the complexity and variability of real-world oscillatory behavior. An improvement would be to manually label oscillatory segments in unaltered time series data. This would increase the reliability of evaluation and model performance, but would, however, be very time-consuming.

The evaluation metrics, such as precision, recall, F1 score, and AUC, are influenced by these challenges. For example, predictions that are counted as false positives may correspond to true, unlabeled oscillations. False positives are usually not something to strive for, but, in this case, they can actually be useful. Since the goal is to detect oscillatory behavior, a detection outside of the labeled segments might still represent valuable information. From this perspective, some false positives are not only acceptable but potentially meaningful.

Another challenge was choosing the POT parameters, particularly the threshold value p. While a natural approach might be to base p on the expected proportion of anomalies, for example, setting $p = 0.99$ under the assumption of approximately 1% anomalies, which proved ineffective. First, the true proportion of anomalies in the test set is unknown, as it contains unlabeled oscillatory content. Second, at $p = 0.99$, the model often fails to identify any ground truth segments as true positives, since other parts of the time series receive higher anomaly scores. Moreover, since POT is fitted on the training data, the resulting threshold may not transfer well to the distribution of the test set. As illustrated in Figure 5.5, applying a fixed threshold across test signals with varying score distributions can lead to poor performance, as it does not adapt to varying score scales.

A more adaptive alternative would be to fit POT separately on the testing scores of each vehicle, allowing the threshold to adapt to each signal's score distribution and reducing the risk of threshold mismatch. This approach is particularly relevant when the model is trained on test cell data, and the training score distribution differs significantly from the diverse test data score distributions. However, it should be

the model that defines what is anomalous, rather than relying on the properties of the test score distribution, highlighting the complexity of threshold selection. As discussed in the previous section, training models based on specific vehicle behaviors could improve performance for those subgroups. In such scenarios, applying POT to training score distributions would be more coherent, as the test score distributions would be more similar within each group. Notably, fitting POT on the training losses was more suitable for the model trained on the field test data, likely because the dataset included a variety of actuator patterns from different vehicles, leading to a more representative threshold.

Segment-level evaluation using IoU provided a clearer measure of how well ranges of oscillatory events were detected. This approach highlighted the model's ability to capture full oscillatory patterns rather than isolated points. However, this evaluation comes with limitations. For instance, if a synthetic oscillation is inserted into a long segment of unlabeled oscillatory behavior, and the model correctly classifies the entire segment as anomalous, it may still not be counted as a true positive, since only the synthetic part is labeled. This leads to an underestimation of the model's performance, despite it capturing the broader oscillatory context accurately.

Moreover, the segment-level evaluation could potentially be improved with additional postprocessing steps. For example, very short predicted segments could be filtered out to avoid false positives, and closely aligned segments could be merged if they appear close in time to better reflect collective anomalies.

6.5 Future Work

There are many possible directions for future work based on the findings in this thesis. In particular, three directions that could lead to improvements or new insights are the continuation and extension of the work in this thesis, the exploration of alternative approaches and the development of more reliable data.

Continuation of This Work

There are several areas of the work presented in this thesis that could be further explored and developed. Based on the results, TranAD appears to generalize better than OC-SVM and likely offers greater potential for improvement. Any continuation of the work in this thesis should therefore focus on advancing TranAD. The optimization of the model was very limited and is, therefore, an obvious area of improvement. This applies to hyperparameter tuning, but experimenting with different window sizes could offer potential gains. According to domain experts, oscillations can vary in both length and frequency, so ideally, the window size should be set dynamically based on what the data looks like. The training procedure could also be enhanced by for example sequence shuffling, to improve the model's ability to generalize.

Furthermore, a valuable extension of TranAD could be to integrate a clustering method as a postprocessing step. Since the model identifies not only oscillations

but also other types of anomalies, it could be useful to categorize the anomalies to make the results easier to interpret and potentially help filter out irrelevant segments.

Alternative Approaches

Beyond improvements to the approach proposed in this thesis, exploring alternative anomaly detection methods could be valuable. Deep learning architectures, such as Recurrent Neural Networks and Autoencoders, remain popular and could be worth exploring further in this context. Additionally, recent research has explored ways to exploit both time- and frequency domain to detect pattern-wise anomalies. Nam *et al.* [72], propose an anomaly detection method using both time domain and frequency domain on time series data. The proposed method is model agnostic and can be applied to any reconstructive model, making it an interesting direction for research. Other possible approaches include shape-based techniques involving, for example, Fourier or wavelet transforms. However, a key limitation with such techniques is the limited knowledge of what oscillations look like in the vehicle data. Many shape-based methods rely on predefined reference patterns, which would require a deeper understanding of the characteristics of oscillations in this context. Any future work in this direction would therefore need to address this challenge as an initial step.

Data Quality

A key limitation in this thesis is the lack of labeled data in combination with the data being contaminated with an unknown proportion of oscillations. Addressing this issue could be done by either creating a dataset with fewer oscillations or collecting labeled data. However, collecting labeled data is often both time-consuming and expensive, especially in a domain like this, requiring expert knowledge. A compromise could be to create a smaller labeled dataset for evaluation or validation during model development. This would make evaluation more accurate and could also enable the use of semi-supervised learning.

Moreover, the challenge of contaminated training data could be addressed to further improve performance. Kim *et al.* [73], propose a model-agnostic, iterative training method that adjusts the importance weights of data samples based on how likely they are to be normal. By assigning higher weights to presumed normal data and lower weights to samples more likely to be anomalous, the method improves the robustness of the model. This approach could improve training reliability and model performance without requiring a fully cleaned or labeled training set.

6.6 Conclusion

This thesis set out to investigate the use of unsupervised anomaly detection techniques for identifying oscillations in high-resolution time series data collected from field test vehicles. In line with the main objectives, two models were developed: one classical machine learning model, One-Class SVM and one more advanced transformer-based model, TranAD. Both were evaluated on their ability to detect

both synthetic and naturally occurring oscillations in real-world actuator data, despite challenges such as the absence of labels and data contamination.

Despite the constraints posed by unlabeled and contaminated data, both models showed the capability to detect oscillations to some extent. However, they also identified other types of anomalies, which, while outside the primary scope of this thesis, could still provide valuable insights. This highlights the broader applicability of the methods, even though isolating oscillatory patterns remains a challenge. TranAD generally performed better than OC-SVM, especially in its ability to generalize across different types of oscillations. However, its performance was shown to be highly dependent on the quality of the training data. Improving the quality of the training data would likely enhance the model's robustness and performance and should be considered for any future work.

As for the broader suitability of unsupervised anomaly detection in this context, the findings suggest that such techniques are indeed a promising starting point for identifying oscillations. Given that they identify anomalies in general, and not only oscillations, future work could incorporate clustering techniques as a postprocessing step to distinguish between different anomaly types more effectively.

Despite these limitations, the models successfully identified oscillatory patterns that had not previously been detected at Volvo Group. By demonstrating that unsupervised methods can identify such patterns in field test data, at least to some extent, this contributes to the development of diagnostic capabilities at Volvo Group.

Bibliography

[1] European Automobile Manufacturers Association. "Fact sheet: CO2 standards for heavy-duty vehicles." (2023), [Online]. Available: https://www.acea.auto/fact/fact-sheet-co2-standards-for-heavy-duty-vehicles. (accessed on: 2025-04-03).

[2] European Commission. "The European Green Deal." (2021), [Online]. Available: https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal_en. (accessed on: 2025-04-03).

[3] Chalmers University of Technology. "Regulations for the use of AI tools in thesis work." (2024), [Online]. Available: https://www.chalmers.se/en/education/your-studies/masters-and-bachelors-thesis/regulations-for-the-use-of-ai-tools/. (accessed on: 2025-02-03).

[4] K. Stouffer, M. Pease, C. Tang, *et al.*, "Guide to Operational Technology (OT) Security," *NIST Special Publication*, vol. NIST SP 800-82r3, Sep. 2022. DOI: 10.6028/NIST.SP.800-82r3.

[5] J. W. Dambros, J. O. Trierweiler, and M. Farenzena, "Oscillation detection in process industries – part i: Review of the detection methods," *Journal of Process Control*, vol. 78, pp. 108–123, Jun. 2019. DOI: 10.1016/j.jprocont.2019.04.002.

[6] T. Hägglund, "A control-loop performance monitor," *Control Engineering Practice*, vol. 3, no. 11, pp. 1543–1551, Nov. 1995. DOI: 10.1016/0967-0661(95)00164-P.

[7] N. F. Thornhill and T. Hägglund, "Detection and diagnosis of oscillation in control loops," *Control Engineering Practice*, vol. 5, no. 10, pp. 1343–1354, Oct. 1997. DOI: 10.1016/S0967-0661(97)00131-7.

[8] K. Zhang, B. Huang, and G. Ji, "Multiple oscillations detection in control loops by using the dft and raleigh distribution," *IFAC-PapersOnLine*, vol. 48, no. 21, pp. 529–534, Oct. 2015. DOI: 10.1016/j.ifacol.2015.09.580.

[9] T. Matsuo, H. Sasaoka, and Y. Yamashita, "Detection and diagnosis of oscillations in process plants," in *Knowledge-Based Intelligent Information and Engineering Systems*, Oxford, UK, 2003, pp. 1258–1264. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-45224-9_170, Accessed on: 2025-03-19.

[10] J. W. Dambros, J. O. Trierweiler, M. Farenzena, and M. Kloft, "Oscillation detection in process industries by a machine learning-based approach," *Indus-*

trial and Engineering Chemistry Research, vol. 58, no. 31, pp. 13 793–14 608, Aug. 2019. DOI: 10.1021/acs.iecr.9b01456.

[11] A. Memarian, S. K. Damarla, B. Huang, Z. Han, and M. Marvan, "Shape-based pattern recognition approaches toward oscillation detection," *Industrial and Engineering Chemistry Research*, vol. 63, no. 9, pp. 4018–4029, Feb. 2024. DOI: 10.1021/acs.iecr.3c03077.

[12] L. Ruff, J. R. Kauffmann, R. A. Vandermeulen, *et al.*, "A unifying review of deep and shallow anomaly detection," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 756–795, May 2021. DOI: 10.1109/JPROC.2021.3052449.

[13] K. Choi, J. Yi, C. Park, and S. Yoon, "Deep learning for anomaly detection in time-series data: Review, analysis, and guidelines," *IEEE Access*, vol. 9, pp. 120 043–120 065, Aug. 2021. DOI: 10.1109/ACCESS.2021.3107975.

[14] S. Schmidl, P. Wenig, and T. Papenbrock, "Anomaly detection in time series: A comprehensive evaluation," *Proceedings of the VLDB Endowment*, vol. 15, no. 9, pp. 1779–1797, May 2022. DOI: 10.14778/3538598.3538602.

[15] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, Jul. 2001. DOI: 10.1162/089976601750264965.

[16] H. Yu, S. Acharya, S. H. H. Ding, and M. Zulkernine, "Pulseanomaly: Unsupervised anomaly detection on avionic platforms with seasonality and trend modeling in transformer networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 2, pp. 1567–1581, Aug. 2025. DOI: 10.1109/TDSC.2024.3446587.

[17] H. Nguyen, A. Hajisafi, A. Abdoli, S. H. Kim, and C. Shahabi, "An evaluation of time-series anomaly detection in computer networks," in *2023 International Conference on Information Networking (ICOIN)*, Bangkok, Thailand, 2023, pp. 104–109. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10049051, Accessed on: 2025-03-19.

[18] M. Thill, W. Konen, H. Wang, and T. Bäck, "Temporal convolutional autoencoder for unsupervised anomaly detection in time series," *Applied Soft Computing*, vol. 112, p. 107 751, Aug. 2021. DOI: 10.1016/j.asoc.2021.107751.

[19] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Anchorage, AK, USA, 2019, pp. 2828–2837. [Online]. Available: https://dl.acm.org/doi/10.1145/3292500.3330672, Accessed on: 2025-03-20.

[20] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "Usad: Unsupervised anomaly detection on multivariate time series," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Virtual Event CA USA, 2020, pp. 3395–3404. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3394486.3403392, Accessed on: 2025-03-20.

[21] A. Iliopoulos, J. Violos, C. Diou, and I. Varlamis, "Detection of anomalies in multivariate time series using ensemble techniques," in *2023 IEEE Ninth International Conference on Big Data Computing Service and Applications*

(BigDataService), Athens, Greece, 2023, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/10233975, Accessed on: 2025-03-20.

[22] J. Song, K. Kim, J. Oh, and S. Cho, "Memto: Memory-guided transformer for multivariate time series anomaly detection," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, New Orleans, LA, USA, 2023, pp. 57 947–57 963. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/3666122.3668647, Accessed on: 2025-03-20.

[23] S. Tuli, G. Casale, and N. R. Jennings, "Tranad: Deep transformer networks for anomaly detection in multivariate time series data," *Proceedings of the VLDB Endowment*, vol. 15, no. 6, pp. 1201–1214, Feb. 2022. DOI: 10.14778/3514061.3514067.

[24] H. Janocha, "Introduction," in *Actuators: Basics and Applications*, H. Janocha, Ed. Heidelberg, Germany: Springer-Verlag Berlin, 2004, ch. 1, pp. 1–17. [Online]. Available: https://doi.org/10.1007/978-3-662-05587-8_1, Accessed on: 2025-03-24.

[25] J. D. Hamilton, *Time Series Analysis*, 2nd ed. Princeton, NJ, USA: Princeton Univ. Press, 2020.

[26] A. Famili, W.-M. Shen, R. Weber, and E. Simoudis, "Data preprocessing and intelligent data analysis," *Intelligent Data Analysis*, vol. 1, no. 1, pp. 3–23, Feb. 1997. DOI: 10.1016/S1088-467X(98)00007-9.

[27] F. T. Lima and V. M. Souza, "A large comparison of normalization methods on time series," *Big Data Research*, vol. 34, Aug. 2023. DOI: 10.1016/j.bdr.2023.100407.

[28] L. Kulanuwat, C. Chantrapornchai, M. Maleewong, *et al.*, "Anomaly detection using a sliding window technique and data imputation with machine learning for hydrological time series," *Water*, vol. 13, no. 13, p. 1862, Jul. 2021. DOI: 10.3390/w13131862.

[29] U. Rebbapragada, P. Protopapas, C. E. Brodley, and C. Alcock, "Finding anomalous periodic time series," *Mach Learn*, vol. 74, pp. 281–313, Dec. 2008. DOI: 10.1007/s10994-008-5093-3.

[30] J. Li, H. Izakian, W. Pedrycz, and I. Jamal, "Clustering-based anomaly detection in multivariate time series data," *Applied Soft Computing*, vol. 100, p. 106 919, Mar. 2021. DOI: 10.1016/j.asoc.2020.106919.

[31] Z.-H. Zhou, *Machine Learning*. Singapore: Springer Nature Singapore, 2021.

[32] T. M. Mitchell, *Machine Learning*. New York, USA: McGraw-Hill, 1997.

[33] J. D. Kelleher, *Deep learning*. Camebridge, England: The MIT Press, 2019.

[34] X. Wu, X. Liu, and Y. Zhou, "Review of unsupervised learning techniques," in *Proceedings of 2021 Chinese Intelligent Systems Conference*, Fuzhou, China, 2021, pp. 576–590, [Online]. Available: https://doi.org/10.1007/978-981-16-6324-6_59. Accessed on: 2025-03-25.

[35] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning: Methods, Systems, Challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Cham, Switzerland: Springer, 2019, ch. 2, pp. 3–33. [Online]. Available: https://doi.org/10.1007/978-3-030-05318-5_1, Accessed on: 2025-03-24.

[36] M. Batra and P. Agrawal, "A survey on hyperparameter optimization of machine learning models," in *2024 2nd International Conference on Disruptive Technologies (ICDT)*, Greater Noida, India, 2024, pp. 11–15, [Online]. Available: https://doi.org/10.1109/ICDT61202.2024.10489732. Accessed on: 2025-03-24.

[37] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, Feb. 2012, ISSN: 15324435.

[38] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Anchorage, AK, USA, 2019, pp. 2623–2631, [Online]. Available: https://doi.org/10.1145/3292500.3330701. Accessed on: 2025-03-25.

[39] S. Shekhar, A. Bansode, and A. Salim, "A comparative study of hyper-parameter optimization tools," in *2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering*, Brisbane, Australia, 2021, pp. 1–6, [Online]. Available: https://doi.org/10.1109/CSDE53843.2021.9718485. Accessed on: 2025-03-25.

[40] S. Hanifi, A. Cammarono, and H. Zare-Behtash, "Advanced hyperparameter optimization of deep learning models for wind power prediction," *Renewable Energy*, vol. 221, Feb. 2024. DOI: 10.1016/j.renene.2023.119700.

[41] V. N. Vapnik, *The Nature of Statistical Learning Theory*, 2nd ed. New York, NY, USA: Springer, 2000, [Online]. Available: https://doi.org/10.1007/978-1-4757-3264-1. Accessed on: 2025-03-19.

[42] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, "A comprehensive survey on support vector machine classification: Applications, challenges and trends," *Neurocomputing*, vol. 408, pp. 189–215, Sep. 2020. DOI: 10.1016/j.neucom.2019.10.118.

[43] J. Ma and S. Perkins, "Time-series novelty detection using one-class support vector machines," in *Proceedings of the International Joint Conference on Neural Networks*, Portland, OR, USA, 2003, pp. 1741–1745, [Online]. Available: https://doi.org/10.1109/IJCNN.2003.1223670. Accessed on: 2025-03-19.

[44] T. Fletcher, "Support vector machines explained," *Tutorial paper*, vol. 1118, pp. 1–19, 2009.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[46] M. N. Nguyen and N. A. Vien, "Scalable and interpretable one-class svms with deep learning and random fourier features," in *Machine Learning and Knowledge Discovery in Databases*, Dublin, Ireland, 2018, pp. 157–172, [Online]. Available: https://doi.org/10.1007/978-3-030-10925-7_10. Accessed on: 2025-03-20.

[47] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: The MIT Press, 2016.

[48] A. Subasi, "Chapter 3 - machine learning techniques," in *Practical Machine Learning for Data Analysis Using Python*, A. Subasi, Ed., Cambridge, MA, USA: Academic Press, 2020, pp. 91–202.

[49] R. Reed and R. J. M. II, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1999.

[50] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, Jul. 2011. DOI: 10.5555/1953048.2021068.

[51] T. Tieleman and G. Hinton, *Lecture 6.5—rmsprop: Divide the gradient by a running average of its recent magnitude*, Coursera: Neural Networks for Machine Learning, Lecture 6, slides 26–31, 2012. [Online]. Available: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Dec. 2014. DOI: 10.48550/arXiv.1412.6980.

[53] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Long Beach, CA, USA, 2017, pp. 6000–6010, [Online]. Available: https://dl.acm.org/doi/10.5555/3295222.3295349. Accessed on: 2025-04-11.

[54] Google, "Attention Is All You Need - Encoder-decoder Architecture", Accessed on: Apr 15, 2025. [Figure]. https://commons.m.wikimedia.org/wiki/File:Attention_Is_All_You_Need_-_Encoder-decoder_Architecture.png.

[55] A. Asadi and R. Safabakhsh, "The encoder-decoder framework and its applications," in *Deep Learning: Concepts and Architectures*, W. Pedrycz and S.-M. Chen, Eds. Cham, Switzerland: Springer Nature, 2020, pp. 133–167.

[56] Google, "Attention Is All You Need - Attention Head", Accessed on: Apr 22, 2025. [Figure]. https://commons.m.wikimedia.org/wiki/File:Attention_Is_All_You_Need_-_Attention_Head.png.

[57] Google, "Attention Is All You Need - Multiheaded Attention", Accessed on: Apr 22, 2025. [Figure]. https://commons.m.wikimedia.org/wiki/File:Attention_Is_All_You_Need_-_Multiheaded_Attention.png.

[58] Y. Lu, Z. Li, D. He, *et al.*, "Understanding and improving transformer from a multi-particle dynamic system point of view," *arXiv preprint arXiv:1906.02762*, Jun. 2019. DOI: 10.48550/arXiv.1906.02762.

[59] J. Guan, Y. Liu, Q. Kong, *et al.*, "Transformer-based autoencoder with id constraint for unsupervised anomalous sound detection," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2023-42, Oct. 2023. DOI: 10.1186/s13636-023-00308-4.

[60] A. Siffer, P. A. Fouque, A. Termier, and C. Largouet, "Anomaly detection in streams with extreme value theory," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Halifax, NS, Canada, 2017, pp. 1067–1075, [Online]. Available: https://doi.org/10.1145/3097983.3098144. Accessed on: 2025-04-17.

[61] G. Naidu, T. Zuva, and E. M. Sibanda, "A review of evaluation metrics in machine learning algorithms," in *Artificial Intelligence Application in Networks*

and Systems: Proceedings of 12th Computer Science On-line Conference, 2023, pp. 15–25, [Online]. Available: https://doi.org/10.1007/978-3-031-35314-7_2. Accessed on: 2025-03-26.

[62] D. Lovell, D. Miller, J. Capra, and A. P. Bradley, "Never mind the metrics-what about the uncertainty? visualising binary confusion matrix metric distributions to put performance in perspective," in *Proceedings of Machine Learning Research*, Honolulu, HI, USA, 2023, pp. 22 702–22 757, [Online]. Available: https://proceedings.mlr.press/v202/lovell23a/lovell23a.pdf. Accessed on: 2025-03-25.

[63] T. Vafeiadis, K. I. Diamantaras, G. Sarigiannidis, and K. C. Chatzisavvas, "A comparison of machine learning techniques for customer churn prediction," *Simulation Modelling Practice and Theory*, vol. 55, pp. 1–9, Jun. 2015. DOI: 10.1016/j.simpat.2015.03.003.

[64] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, Jun. 2006. DOI: 10.1016/j.patrec.2005.10.010.

[65] EpochFail, "Threshold roc", Accessed on: Mar 26, 2025. [Graph]. https://commons.wikimedia.org/wiki/File:Threshold_roc.wikipedia_edit.svg.

[66] T. J. Lee, J. Gottschlich, N. Tatbul, E. Metcalf, and S. Zdonik, "Precision and recall for range-based anomaly detection," in *Proceedings of the SysML Conference*, Stanford, CA, USA, 2018, [Online]. Available: https://mlsys.org/Conferences/doc/2018/70.pdf. Accessed on: 2025-05-08.

[67] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA, 2019, pp. 658–666, [Online]. Available: https://ieeexplore.ieee.org/document/8953982. Accessed on: 2025-04-03.

[68] S. Tuli, G. Casale, and N. R. Jennings. "TranAD." (2022), [Online]. Available: https://github.com/imperial-qore/TranAD. (accessed on: 2025-05-05).

[69] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019, pp. 8026–8037, [Online]. Available: https://dl.acm.org/doi/10.5555/3454287.3455008. Accessed on: 2025-05-05.

[70] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019*, New Orleans, LA, USA, 2019, [Online]. Available: https://doi.org/10.48550/arXiv.1711.05101. Accessed on: 2025-05-19.

[71] L. Ruff, R. Vandermeulen, N. Goernitz, *et al.*, "Deep one-class classification," in *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden, 2018, pp. 4393–4402, [Online]. Available: https://proceedings.mlr.press/v80/ruff18a.html. Accessed on: 2025-03-31.

[72] Y. Nam, S. Yoon, Y. Shin, *et al.*, "Breaking the time-frequency granularity discrepancy in time-series anomaly detection," in *Proceedings of the ACM Web Conference 2024*, Singapore, Singapore, 2024, pp. 4204–4215, [Online].

Available: https://doi.org/10.1145/3589334.3645556. Accessed on: 2025-05-19.

[73] M. Kim, J. Yu, J. Kim, T. H. Oh, and J. K. Choi, "An iterative method for unsupervised robust anomaly detection under data contamination," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 10, pp. 13 327–13 339, May 2023. DOI: 10.1109/TNNLS.2023.3267028.

A

Appendix: Training and Validation Loss

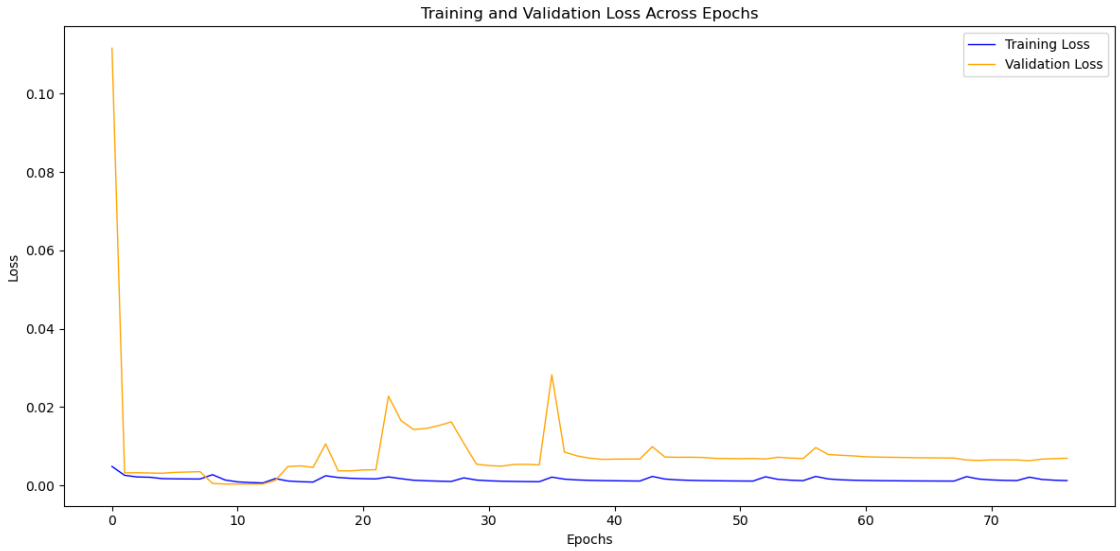


Figure A.1: Training and validation loss for TranAD trained on univariate field test data. The blue line shows training loss, while the orange line shows validation loss. Peaks in validation loss correspond to transitions between vehicles in the training sequence.

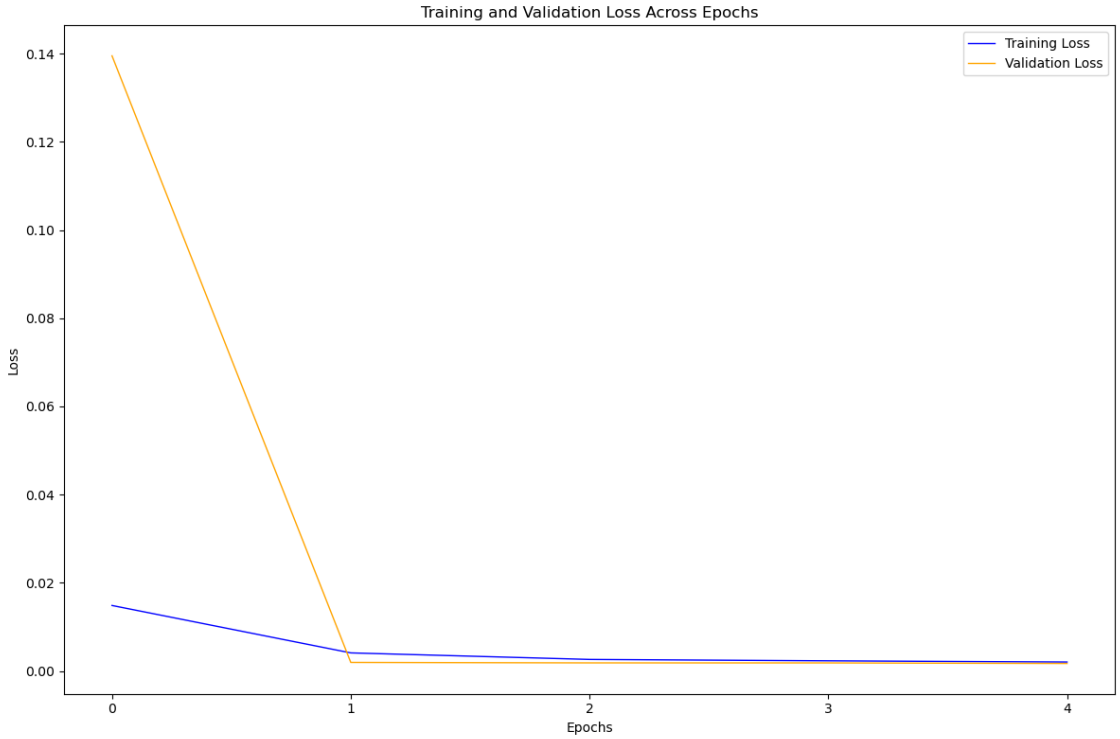


Figure A.2: Training and validation loss for TranAD trained on univariate test cell data. The blue line shows training loss, while the orange line shows validation loss. Small number of epochs due to early stopping.

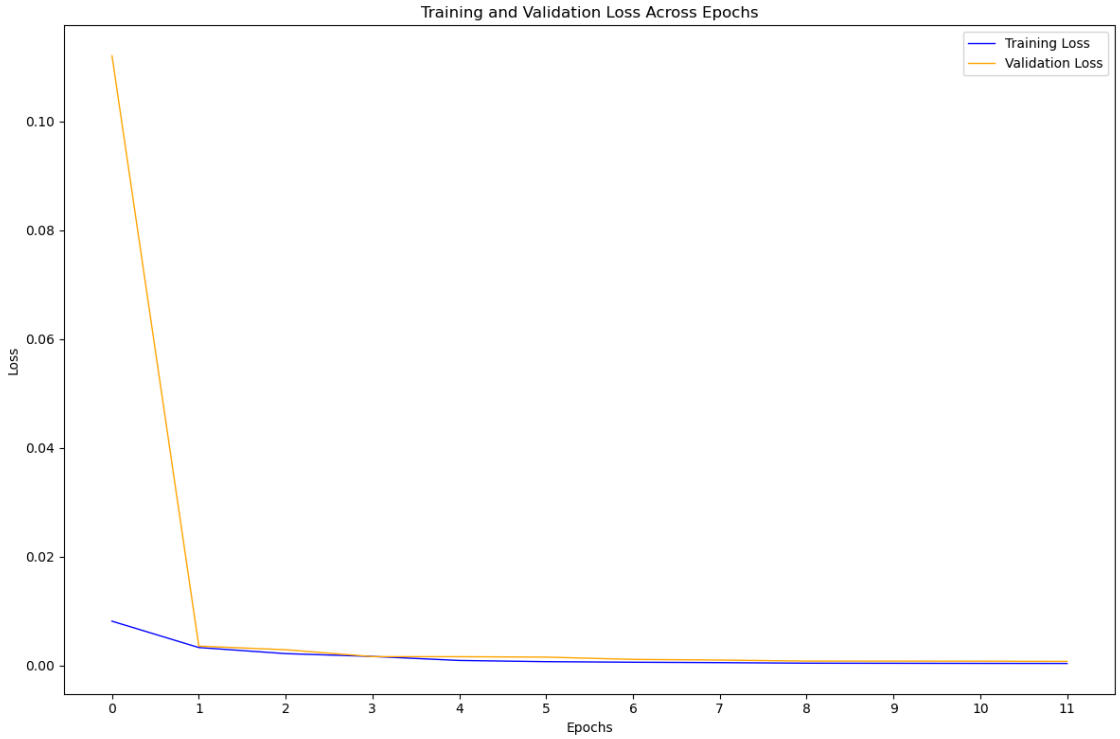


Figure A.3: Training and validation loss for TranAD trained on multivariate test cell data. The blue line shows training loss, while the orange line shows validation loss.

B

Appendix: Performance Results TranAD

Table B.1: Window-level performance of TranAD, with threshold from POT with parameters $p = 0.90$ and $q = 10^{-5}$, over the two test set variants across the four vehicle time series. Best results are shown in bold.

Test Set Variants	Vehicle	Precision	Recall	F1-score	AUC
1% manually discovered oscillations	1	0.0146	0.3112	0.0279	0.5553
	2	0.0168	**0.3285**	0.0320	0.5340
	3	0.0142	0.3083	0.0271	0.5461
	4	0.0457	0.3076	0.0795	**0.6338**
0.5% manually discovered, 0.5% synthetic oscillations	1	0.0162	0.3203	0.0308	0.5682
	2	0.0148	0.3290	0.0283	0.5440
	3	0.0157	0.3307	0.0300	0.5604
	4	**0.0650**	0.2974	**0.1067**	0.6251

Table B.2: Performance of TranAD, with threshold from POT with parameters $p = 0.90$ and $q = 10^{-5}$, over the two test set variants across the four vehicle time series. P_X and R_X denote range-based precision and recall, computed from point-level labels. TP, FN, and FP are segment-level counts. The best range-based results are shown in bold.

Test Set Variants	Vehicle	P_X	R_X	TP	FN	FP
1% manually discovered oscillations	1	0.0198	0.2230	43	195	14,571
	2	0.0241	**0.2545**	19	77	4,955
	3	0.0323	0.2045	86	361	15,101
	4	0.0677	0.1850	14	79	1,584
0.5% manually discovered, 0.5% synthetic oscillations	1	0.0208	0.2396	80	236	14,527
	2	0.0195	0.2525	25	84	4946
	3	0.0396	0.2089	132	457	15,093
	4	**0.0822**	0.1723	28	134	1,563