

Anomaly Detection in the Era of Industry 4.0: Generative Models for Multivariate Time Series

Maximilian Hoh

Vollständiger Abdruck der von der Fakultät für Elektrische Energiesysteme und
Informationstechnik der Universität der Bundeswehr München zur Erlangung des
akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

angenommenen Dissertation.

Gutachter: Prof. Dr. Christoph Kutter
Prof. Dr. Alfred Schöttl

Die Dissertation wurde am 20.11.2025 bei der Universität der Bundeswehr München
eingereicht und durch die Fakultät für Elektrische Energiesysteme und
Informationstechnik am 09.07.2026 angenommen. Die mündliche Prüfung fand am
17.07.2026 statt.

*”Making everything more efficient should
make everybody happier.”*

Geoffrey Hinton

ABSTRACT

Time series anomaly detection is a challenging task and is applicable in many scenarios. Especially in industrial environments, anomaly detection is widely underexplored due to the combination of individual applications and high-frequency operation. The often multivariate nature and the contextual and temporal dependencies of time series data require modern models that can identify novel patterns from historical data.

This work primarily explores whether novel architectures utilizing generative models and modern deep learning techniques can reliably detect anomalies in industrial environments in real time. While current deep learning approaches fall short of investigating the model's performance under varying input window sizes, we believe that a scalable model benefits from broader context windows, leading to better detection results. We design different novel architectures that leverage generative adversarial networks, the transformer architecture, and diffusion models. We focus on reconstruction-based approaches trained solely on nominal data. Anomalous patterns cannot be accurately reconstructed during inference, leading to high anomaly scores. The scoring mechanism is thereby not only restricted to the difference between the reconstructed and the original signal, but can also incorporate other metrics, such as similarity measurements.

Benchmark datasets are essential for new methods, as they make different models and configurations comparable. While multiple datasets for multivariate time-series anomaly detection exist, we find that only a few are representative and can be used to evaluate different models with large input windows. We evaluate our and other often-used models on a subset of those datasets and investigate their performance across different window sizes, thereby showing that current approaches lack scalability and that their performance is misleading.

The path from anomaly score to detection result is by no means standardized, which impairs the comparability of algorithms. We discuss different thresholding techniques and result calculations in this thesis and compare them using our and various other state-of-the-art detection algorithms on different datasets. In particular, our novel transformer-based architecture demonstrates scalability, low latency, and robust detection performance in our experiments, enabling its use in production environments.

The simulated deployment in two distinct real-world industrial use cases demonstrates the efficiency of our approach. Because long-lived machinery rarely exhibits anomalies within a constrained monitoring period, we generate synthetic anomalies and demonstrate our model's capabilities.

KURZFASSUNG

Die Erkennung von Anomalien in Zeitreihen ist eine komplexe Aufgabe, die in vielen Anwendungsbereichen zum Einsatz kommt. Insbesondere in industriellen Umgebungen wird die Anomalieerkennung aufgrund der Kombination aus individuellen Anwendungen und hochfrequentem Betrieb noch viel zu wenig genutzt. Aufgrund der oft multivariaten Beschaffenheit sowie der kontextuellen und zeitlichen Abhängigkeiten der Signale sind moderne Modelle erforderlich, die auf Grundlage früherer Daten neue Muster und damit anomales Verhalten erkennen können.

Diese Dissertation untersucht in erster Linie, ob neuartige Architekturen Anomalien in industriellen Umgebungen zuverlässig in Echtzeit erkennen können. Während aktuelle Deep-Learning-Ansätze die Leistung des Modells unter variierenden Eingabefenstergrößen nicht ausreichend untersuchen, glauben wir, dass ein skalierbares Modell von größeren Kontextfenstern profitieren kann. Wir entwerfen neuartige Architekturen, die generative adversarische Netzwerke, die Transformer Architektur und Diffusionsmodelle nutzen. Wir konzentrieren uns auf rekonstruktionsbasierte Ansätze, die ausschließlich auf nominalen Daten trainiert werden und während der Inferenz anomale Muster nicht rekonstruieren können, was zu hohen Anomaliewerten führt. Der Bewertungsmechanismus beschränkt sich nicht nur auf die Differenz zwischen dem rekonstruierten und dem ursprünglichen Signal, sondern kann auch andere Metriken wie Ähnlichkeitsmessungen einbeziehen.

Benchmark-Datensätze sind essenziell für neue Verfahren, da sie verschiedene Modelle und Konfigurationen miteinander vergleichbar machen. Es gibt mehrere Datensätze für die Zeitreihen-Anomalieerkennung, wobei nur wenige repräsentativ sind und sich zur Bewertung verschiedener Modelle mit großen Eingabefenstern eignen. Wir evaluieren unsere und andere häufig verwendete Modelle anhand einer Teilmenge dieser Datensätze und untersuchen ihre Leistung bei verschiedenen Fenstergrößen. Dabei zeigen wir, dass es den derzeitigen Ansätzen an Skalierbarkeit fehlt und ihre Leistung teilweise beschönigt wird.

Der Weg vom Anomaliewert zum Erkennungsergebnis ist nicht standardisiert, was die Vergleichbarkeit von Algorithmen beeinträchtigt. In dieser Arbeit werden verschiedene Schwellenwerttechniken und Ergebnisberechnungen diskutiert und anhand unserer und anderer moderner Algorithmen an verschiedenen Datensätzen verglichen. Insbesondere unsere neuartige transformerbasierte Architektur zeigt in den Experimenten Skalierbarkeit, geringe Latenz und robuste Erkennungsleistung, sodass sie in Produktionsumgebungen eingesetzt werden kann.

Der simulierte Einsatz in zwei unterschiedlichen realen industriellen Anwendungsfällen zeigt die Effizienz unseres Ansatzes. Da langlebige Maschinen innerhalb eines begrenzten Überwachungszeitraums selten Anomalien aufweisen, generieren wir synthetische Anomalien und demonstrieren so die Fähigkeiten unseres Modells.

ACKNOWLEDGMENTS

This thesis was created during my work as a research associate at the University of Applied Sciences Munich. I want to take this opportunity to thank everyone who has accompanied and supported me over the past years:

Alfred, attending your lecture in the winter semester of 2017/2018 was my starting point in the world of *machine learning*, and since then, you have been my first point of contact and mentor in this field. Thank you for letting me work at the university under your guidance, for the trust you placed in me, and for the freedom you gave me. Your perspective, patience, and engagement have not only made this thesis possible but also helped me grow both as a researcher and as a person.

I want to express my special thanks to Prof. Dr. Christoph Kutter for supervising this work, for his feedback, and for the trust he has placed in me.

Thank you to my colleagues and friends for the teamwork, inspiring discussions, and shared coffee breaks that made this journey more enjoyable. Although I would describe us as more lone fighters, we complemented each other professionally. We had a strong personal connection that went far beyond a collegial working relationship, which made it easier to come to the lab every day.

To my family, thank you for always believing in me and always offering a sympathetic ear to me and my concerns. Even when I missed countless family moments, you stayed supportive and never stopped cheering me on. I have enjoyable memories of the *brothers' weekends*, when I was able to switch off and escape from the world of science.

Finally, Francesca — thank you for your endless love, patience, and understanding. You have been my greatest source of comfort and strength. Thank you for always being there, even when I wasn't easy to reach or when my focus was elsewhere. You believed in me long before I believed in myself, and that faith carried me through moments of doubt. Without you, this work would not have been possible.

CONTENTS

Abstract	iii
Kurzfassung	v
Acknowledgments	vii
List of Figures	xiii
List of Tables	xvii
Abbreviations	xxi
1 Introduction	1
1.1 Motivation	2
1.2 Research Objectives and Contributions	3
1.3 Structure of the Thesis	4
2 Background	7
2.1 Time Series Definition	7
2.2 Anomaly Detection	8
2.2.1 Anomaly Types	9
2.2.2 Detection Tasks	12
2.3 Underlying Machine Learning Concepts	13
2.3.1 Generative Adversarial Network	14
2.3.2 Encoder-Decoder Structures	15
2.3.3 Diffusion Models	21
2.3.4 Benefits and Drawbacks of the discussed Vanilla Architectures	23
2.4 Summary	25
3 Traditional and Current Methods for Anomaly Detection	27
3.1 Early Works	28
3.2 Generative Models	30
3.2.1 GAN-based Models	31
3.2.2 Transformer-based Models	34
3.2.3 Diffusion-based Models	40

3.3	Summary	43
4	Novel Generative Deep Learning Models for Time Series Anomaly Detection	47
4.1	General Setup for Training and Evaluating Models	49
4.2	Generative Adversarial Networks with Stabilized Training	49
4.2.1	Eliminating Disadvantages of GANs	50
4.2.2	Conditional GAN for Anomaly Detection	52
4.3	Scalable Transformer Model for Large Input Windows	59
4.3.1	Improving Attention Calculation to Reduce Complexity	60
4.3.2	Transformer U-Net for Anomaly Detection	65
4.3.3	Hyperparameter Settings	68
4.4	Diffusion Model	69
4.4.1	Forward Process	70
4.4.2	Conditioned Backward Process	75
4.4.3	Experimental Studies	79
4.4.4	Summary	84
4.5	Experimental Comparison of the Introduced Models	85
4.6	Summary	87
5	Evaluation Strategies	89
5.1	Analysis of Anomaly Scoring and Thresholding Techniques	90
5.2	Relevant Performance Metrics	94
5.3	Summary	97
6	Case Study: Publicly Available Datasets	99
6.1	Data Analysis	100
6.2	Experimental Results	113
6.2.1	Results	114
6.2.2	Ablation Studies	117
6.2.3	Results of Baseline Approaches	120
6.3	Summary and Conclusion	121
7	Case Study: Real-World Industrial Data	127
7.1	Setup and Objective	128
7.2	Data Inspection	130
7.3	Anomaly Injection	134
7.4	Experimental Results	135
7.5	Summary and Conclusion	139

8 Summary, Conclusion and Outlook	141
8.1 Limitations	143
8.2 Outlook of Time Series Anomaly Detection	143
8.3 Final Remarks	144
Bibliography	147
Curriculum Vitae	167
A Public Datasets Details	169
A.1 Tabular Per-Entity Overview	169
A.2 Additional Dataset Plots	169
A.2.1 Value Ranges	169
A.2.2 Dataset Examples	170
A.3 Dataset Results	170
B Industrial Dataset Details	179
B.1 KIWA	179
B.2 DigiSeal	180

LIST OF FIGURES

2.1	Synthetic univariate time series with seasonality and trend.	8
2.2	Overview of different types of anomalies.	9
2.3	Synthetic univariate time series with different point and collective anomalies.	10
2.4	Simplified structure of a vanilla GAN	14
2.5	Simplified structure of a vanilla autoencoder.	16
2.6	Simplified structure of a U-Net.	17
2.7	Simplified structure of a variational autoencoder.	18
2.8	Structure of a vanilla Transformer.	21
2.9	Schematic visualization of forward and backward process of a diffusion model.	22
3.1	Simplified architectures of different GAN-based models.	32
3.2	Simplified structures of different transformer-based architectures.	35
3.3	Different extensions or adaptations of the standard attention mechanism.	36
3.4	Attention module of AnomalyTransformer.	39
3.5	Example of conditioning channel $c = 0$ of input $\mathbf{X}$ with DiffAD.	41
3.6	Example of conditioning channel $c = 0$ of input $\mathbf{X}$ with ImDiffusion.	42
3.7	Example of conditioning channel $c = 0$ of input $\mathbf{X}$ with DDMT.	43
4.1	Snippet of the training data.	48
4.2	Conditional generator as an autoencoder.	53
4.3	Conditional generator as an autoencoder with a second decoder.	56
4.4	Impact of different sparsity regularization techniques on the adversarial loss of discriminator and generator.	57
4.5	Impact of different sparsity regularization techniques on the reconstruction loss of discriminator and generator.	58
4.6	Reconstruction loss curves of different runs with the Dual-AE-GAN setup.	59
4.7	Comparison of attention algorithms regarding time (left) and memory (right) consumption.	65
4.8	Overview of the Swin1D-AD architecture.	66
4.9	Impact of the model dimension and input window size on its reconstruction capabilities. The color code indicates the value of the MSE and is shared between the two sub-figures.	69

List of Figures

4.10	The values of $\sqrt{\alpha}$ with different schedulers.	70
4.11	Distribution of k in epoch 50 during training when applying the adaptive non-uniform sampling strategy and $K = 100$ in contrast to uniform sampling, indicated in red.	73
4.12	Visualization of the mean values and standard deviations of different training samples when applying the DDPM forward step at different diffusion steps. The dashed lines represent the target distribution at $k = K$	75
4.13	Original sample and the reconstruction of the simple autoencoder.	77
4.14	U-Net structure used in the diffusion model.	78
4.15	Impact of different configurations on the training process represented by training losses (as MSE) over 50 epochs of training with respect to the base implementation.	80
4.16	Reconstruction error for each diffusion sampling step of arbitrary test samples with and without anomalies.	83
4.17	Reconstruction of a single channel for each diffusion sampling step of arbitrary test samples with and without anomalies.	84
5.1	Top: A segment from the first channel of entity machine-1-6 of SMD and the shifted (reconstructed) version. Bottom: The corresponding anomaly score calculated as the MSE.	90
5.2	Top: A segment from the first channel of entity machine-1-6 of SMD and the reconstructed signal. Bottom: The corresponding anomaly score for the displayed channel, as well as the mean anomaly score across all channels.	92
5.3	Anomaly scores of entity machine-1-6 of SMD along with different thresholds.	94
5.4	Anomaly scores of the first channel of entity machine-1-6 of SMD along with different thresholds.	95
5.5	Visualization of the point adjustment strategy.	96
6.1	The first channel of entity P-3 of the SMAP dataset (left) with the corresponding value distribution (right). The top row visualizes the training data and the bottom row the testing data, where the labeled anomaly is highlighted.	102
6.2	A subsequence of the first channel of entity P-3 of the SMAP dataset.	103
6.3	Nominal data distribution for channel 0 of different entities of the SMAP dataset.	104
6.4	Nominal data distribution for channel 0 of different entities of the MSL dataset.	104

6.5	Autocorrelation function with 95% confidence band. Only the first channel is analyzed.	105
6.6	Analysis of the anomalies in the SMAP test dataset across all entities.	105
6.7	Analysis of the anomalies in the MSL test dataset across all entities.	106
6.8	Nominal data distribution for selected channels of entity machine-3-11 of the SMD dataset	106
6.9	Feature correlation of different entities from the SMD training dataset.	107
6.10	Analysis of the anomalies in the SMD test dataset across all entities.	107
6.11	Nominal data distribution for selected channels of the PSM dataset.	108
6.12	Feature and time correlation of the PSM training dataset.	108
6.13	Analysis of the anomalies in the PSM test dataset.	109
6.14	Nominal data distribution for selected channels of the SWaT dataset.	110
6.15	Nominal data distribution for selected channels of the WADI dataset.	110
6.16	Feature correlation of the SWaT training dataset	110
6.17	Analysis of the anomalies in the SWaT test dataset.	111
6.18	Analysis of the anomalies in the WADI test dataset.	111
6.19	Example training progress and results of Swin1D-AD when trained on entity machine-1-6 from SMD.	114
6.20	Comparison of the reconstructions on SMD machine-1-1 (without anomalies).	117
6.21	Comparison of the reconstructions on SMD machine-1-1 (with anomalies).	118
7.1	The process of ejecting a die from a wafer to a substrate.	128
7.2	Schematic sensor setup of DigiSeal. Different colors indicate the different components: machine, seal, bearing, and suspension with rotation axis. The flow of the fluid is visualized with arrows.	130
7.3	Autocorrelation functions of each channel of the three-axis acceleration sensor.	131
7.4	Overlay of multiple punch-out processes in the KIWA dataset.	132
7.5	Feature correlation of the DigiSeal dataset (one day).	134
7.6	Absolute differences of monitored temperatures of sensor_setup2. Top: Differences of inlet and outlet temperatures. Bottom: Differences at DE and NDE.	135
7.7	KIWA: Synthetical anomalies and their reconstructions by Swin1D-AD ₅₁₂ . Anomalies are in the highlighted areas.	137
7.8	DigiSeal: Synthetical anomalies and their reconstructions by Swin1D-AD ₅₁₂ . Anomalies are in the highlighted areas.	138

List of Figures

- A.1 Channel-wise value range of nominal data of different datasets. Diamonds represent constant values. 171
- A.2 Plots of machine-3-11 from SMD: (a) Segment of the fourth channel of test data. The highlighted area is the labeled anomaly. (b) Feature correlations of the training data. 173

LIST OF TABLES

3.1	Overview of presented GAN-based algorithms and their particular modifications.	34
3.2	Overview of presented transformer-based algorithms for time series data and their particular modifications.	40
3.3	Overview of presented DDPM-based algorithms for time series anomaly detection and their particular modifications.	43
3.4	Input window sizes of presented time series anomaly detection algorithms.	44
4.1	Additional hyperparameters for different sparsity calculations.	55
4.2	Overview of the different GAN implementations	56
4.3	Comparison of discussed attention mechanisms.	64
4.4	MSE of normal and anomalous time steps in the testing data. a) model prediction, b) β scheduler, c) dimension of the condition, d) varying K , e) self-conditioning, f) k -sampling.	82
4.5	Effect of an early stopping strategy on sample generation.	83
4.6	Comparison of reconstruction errors of different models with varying input window sizes.	86
4.7	Comparison of inference speed in reconstructed samples per second for each model with varying input window sizes. The values displayed are given in Hertz (Hz).	86
4.8	Comparison of inference speed in reconstructed time steps per second for each model with varying input window sizes. The values displayed are given in Kilohertz (kHz).	87
5.1	Anomaly score calculations of different approaches besides the standard reconstruction error in the time domain.	91
5.2	Example calculation of different metrics.	96
6.1	Overview of benchmark datasets for time series anomaly detection datasets.	100
6.2	Number of time steps for training and testing data of different datasets along with the anomaly ratio.	101
6.3	Strengths and weaknesses of publicly available datasets.	112

6.4	Evaluation on two entities of SMD with different methods of F1-score calculations and varying threshold algorithms. The scores represent the mean F1-scores across different input window sizes with the corresponding standard deviation. Best point-wise F1-scores for every non-iterative thresholding algorithm are highlighted in bold.	123
6.5	Evaluation of our algorithms on two entities of SMD with varying threshold algorithms. AUROC and AUPRC scores are shown here. Best scores for every non-iterative thresholding algorithm are highlighted in bold.	124
6.6	Resulting F1-scores of different ablation studies discussed. Green , black, and red indicate performance relative to the standard models, listed as <i>baseline</i> here. Experiments were conducted on SMD machine-1-6. The best score of each adjustment is highlighted in bold.	124
6.7	Evaluation on two entities of SMD with different baseline methods of F1-score calculations and varying threshold algorithms. The scores represent the mean F1-scores across different input window sizes with the corresponding standard deviation. Best point-wise F1-scores for every non-iterative thresholding algorithm are highlighted in bold.	125
7.1	Mean reconstruction error during validation of real-world datasets (normal data only).	136
7.2	Mean inference times of Swin1D-AD ₅₁₂ in samples per second on real-world datasets.	139
A.1	Entities of the MSL dataset.	170
A.2	Entitites of the SMAP dataset.	172
A.3	Overview of the entities of SMD.	173
A.4	Evaluation on SMD machine-1-1 with different methods of F1-score calculations and varying threshold algorithms. Best point-wise F1-scores for every non-iterative thresholding algorithm are highlighted in bold.	174
A.5	Evaluation of different algorithms on SMD machine-1-1 with varying threshold algorithms. AUROC and Area under the P-R Curve (AUPRC) scores are shown here. Best scores for every non-iterative thresholding algorithm are highlighted in bold.	175
A.6	Evaluation on SMD machine-1-6 with different methods of F1-score calculations and varying threshold algorithms. Best point-wise F1-scores for every non-iterative thresholding algorithm are highlighted bold.	176
A.7	Evaluation of different algorithms on SMD machine-1-6 with varying threshold algorithms. AUROC and AUPRC scores are shown here. Best scores for every non-iterative thresholding algorithm are highlighted in bold.	177

B.1	Channel definitions of the KIWA dataset.	179
B.2	Channel definitions of sensor_setup1 of the DigiSeal dataset.	180
B.3	Channel definitions of sensor_setup2 of the DigiSeal dataset.	180

ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
ARIMA	Autoregressive Integrated Moving Average
AUPRC	Area under the P-R Curve
AUROC	Area under the ROC Curve
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DDIM	Denoising Diffusion Implicit Model
DDPM	Denoising Diffusion Probabilistic Model
DE	drive end
DFT	Discrete Fourier Transformation
DTW	Dynamic Time Warping
ELBO	Evidence Lower Bound
ELU	Exponential Linear Unit
FFT	Fast Fourier Transformation
FN	false negative
FP	false positive
FPR	false positive rate
GAN	Generative Adversarial Network

Abbreviations

GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
ISA	Incident Surprise Anomaly
KL	Kullback-Leibler
kNN	k -Nearest Neighbors
LLM	Large Language Model
LN	Layer Normalization
LOESS	locally estimated scatterplot smoothing
LOF	Local Outlier Factor
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MHA	Multi-Head Attention
MLP	Multilayer Perceptron
MSE	Mean Squared Error
MSL	Mars Science Laboratory rover
NASA	National Aeronautics and Space Administration
NDE	non-drive end
NLP	Natural Language Processing
NTP	Network Time Protocol
OOD	Out-of-Distribution
P	Precision
PCA	Principle Component Analysis
POT	Peaks Over Threshold
PTP	Precision Time Protocol

PSM	Pooled Server Metrics
R	Recall
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SMAP	Soil Moisture Active Passive
SMD	Server Machine Dataset
SNR	signal-to-noise ratio
SOTA	state-of-the-art
SRN	Sparse Regularization Network
STB	Swin Transformer Block
STL	Seasonal-Trend Decomposition using LOESS
SVD	Singular Value Decomposition
SWaT	Secure Water Treatment
Swin	Shifted window
TN	true negative
TP	true positive
TPR	true positive rate
VAE	Variational Autoencoder
WADI	Water Distribution
WGAN	Wasserstein GAN

1

INTRODUCTION

The fourth industrial revolution is defined by the ongoing enhancement of knowledge and control over production processes. The concept of *Industry 4.0*, introduced in Germany in 2011, represents a political vision that incorporates various techniques [1]. This includes connecting industrial facilities, collecting and analyzing data, and automating processes. Modern manufacturing plants are increasingly equipped with sensors that capture a variety of data streams in real-time. These sensors, ranging from vibration and acoustic to temperature, pressure, and visual, are a key component of smart factories, offering a better understanding of the condition, behavior, and performance of industrial assets [2, 3, 4].

A principle of Industry 4.0 is smart manufacturing, which relies on the seamless integration of data from those sensors to enhance productivity, quality, and resilience [5]. A valuable tool of this transformation is the usage of Artificial Intelligence (AI), which enables factories to make data-driven decisions, identify inefficiencies, and predict failures before they occur. Among its core tasks in this domain is anomaly detection, which involves identifying data patterns that deviate from expected norms [6].

Industrial time series data, often characterized by its multivariate nature, high frequency, and non-stationarity, presents unique challenges for anomaly detection. Espe-

cially when monitoring movements of machines or their components with high sampling rates, the data can accumulate quickly to a hard-to-manage scale. Time series datasets can include millions of data points daily for each machine with complex spatial and temporal dependencies [7]. Traditional threshold-based monitoring systems often fall short in finding structures in data, leading to false alarms or missed detections. To tackle this, machine learning techniques, including deep learning, have emerged as effective alternatives, capable of capturing nonlinear temporal dynamics and subtle variations [8]. Due to the recent success of generative models across various applications, such as text, image, and video generation, there has been a new level of attention towards AI-based use-cases across different domains, including time series anomaly detection.

The subsequent sections will further examine the motivations for this research and outline the key challenges to be addressed for efficient and effective real-world anomaly detection in industrial environments.

1.1 Motivation

There is an increasing trend in the usage of sensor technologies and AI-based systems in modern manufacturing, but there are still limitations and challenges concerning the reliability, scalability, and interpretability of current anomaly detection techniques. Due to the complexity and volume of industrial time series data, traditionally used statistical and rule-based methods often fall short in identifying anomalies in unseen data accurately. These approaches typically rely on predefined thresholds that trigger alarms or on simple assumptions. While these concepts may be a good starting point, they are unsuitable for identifying context-specific anomalies or interpreting complex patterns with temporal and spatial dependencies [7].

Machine learning techniques are more flexible but still encounter challenges in time series anomaly detection. The lack of labels and the rare appearance of anomalies prevent the usage of supervised methods, such as classifiers, and the multi-dimensional and high-frequency nature of the data results in unsuitable inference times. Moreover, many existing models operate as black boxes, providing little transparency or actionable insight to domain experts tasked with maintaining critical systems [6].

Despite these challenges, manufacturers that integrate a robust anomaly detection system into their process have the potential to draw substantial benefits from it. The advantages include a step towards predictive maintenance, reduced facility downtime, quality control, enhanced safety, and increased efficiency. All of those contribute to a potential competitive edge over rival companies, as operating costs can be reduced. For instance, the early identification of wear and tear on moving components or the detection of leaks in seals enables the prevention of breakdowns and unwanted incidents. When an anomaly

is detected that indicates a degradation of a component, proactive actions can be initiated to extend its remaining lifetime or to prepare for its replacement. Additionally, advanced anomaly detection methods can facilitate autonomous decision-making, enabling systems to adapt and recover from disturbances without requiring human intervention.

The growing interest in generative models and the increasing success in addressing anomaly detection problems show the demand for advanced deep learning techniques that can precisely capture the normal behavior of industrial systems and thereby identify anomalies in an unsupervised and interpretable fashion. This highlights the need for a deeper investigation into how modern architectures and generative models can be designed and utilized to fill the gap between theoretical performance and practical utility in real-world industrial anomaly detection.

1.2 Research Objectives and Contributions

The research questions (RQ) of this thesis can be formulated as follows:

RQ1: What strategies can be developed to overcome the scarcity of labeled anomaly data in industrial environments?

Real-world sensor data is primarily unlabeled, and anomalies occur rarely. Modern approaches, therefore, rely on unsupervised training of reconstruction-based models. The trained models are not able to reconstruct anomalous time steps or subsequences, leading to high reconstruction errors. This study employs and develops various unsupervised generative models and demonstrates their capabilities.

RQ2: How can generative models be effectively applied to detect anomalies in high-dimensional, multivariate industrial time series data?

Validation of the developed models on both publicly available datasets for time series anomaly detection and real-world data gathered from industrial facilities can help to demonstrate their practical applicability. The datasets are multivariate and are partly derived from sensors with high sampling rates.

RQ3: To what extent can generative models outperform traditional techniques in terms of detection accuracy, robustness, and adaptability in real-world industrial settings?

The novel models are compared with traditional approaches using standardized metrics and vivid examples.

RQ4: How can anomaly detection systems based on generative models be integrated into smart manufacturing pipelines to enable real-time, autonomous decision-making?

The industrial anomaly detection framework includes sensor setups, data-collection methods, and an alert system that indicates when a model has detected an anomaly. The use of specific machine learning operation tools enables easy model versioning and deployment. We show the inference latency of our chosen model to underline its applicability in real-world scenarios.

1.3 Structure of the Thesis

This thesis is structured as follows:

- Chapter 2 provides an overview of the foundational concepts relevant to this thesis. It serves as an introduction and overview to the domain of time series analysis, anomaly detection and modern generative deep learning techniques. The focus is thereby on topic-related definitions and mathematical explanations without going into excessive technical detail.
- Chapter 3 presents different methods for time series analysis. Traditional methods are discussed and categorized, serving as the basis for more complex deep learning-based approaches that heavily rely on generative models and modern architectures. This chapter investigates the concepts state-of-the-art (SOTA) models that rely on structures presented earlier.
- Chapter 4 introduces novel techniques for time series anomaly detection, building on the insights gained from previously discussed generative models. Their structure and relevant design choices are explained in detail along with their advantages and disadvantages. Further, we compare these innovative models by training them on an illustrative time series.
- Chapter 5 discusses the steps from the model output towards effectively evaluating and comparing different algorithms. It introduces various metrics designed to determine the quality and performance of these models. We focus on metrics that are widely used and applied to anomaly detection problems.
- Chapter 6 lists commonly used publicly available datasets for multivariate time series anomaly detection. In addition to a statistical analysis, it discusses the strengths and weaknesses of each dataset. The developed models are trained on a subset of these datasets and the resulting scores are further put in context.
- Chapter 7 focuses on real-world data obtained from industrial facilities and machines. After describing the datasets and analyzing the data, the models' outcomes are presented.
- Chapter 8 concludes this thesis by summarizing its key findings in the context of the research objectives defined above. We also offer a vision for potential future developments in the field of time series anomaly detection.

In this thesis, the following mathematical notation is used: Matrices are represented by uppercase bold letters, such as $\mathbf{X}$, and vectors by lowercase bold letters, such as $\mathbf{x}$. The superscript $\top$ is used to indicate the transpose of a matrix or vector so that $\mathbf{x}^\top$ is the row vector $(x_1, \dots, x_T)$ of length T .

2

BACKGROUND

This chapter gives an introduction to the background of this thesis: Section 2.1 briefly outlines the mathematical definition of a time series, while Section 2.2 defines *anomaly detection* in a time series context. In modern approaches, anomalies are detected using advanced machine learning techniques. Section 2.3 introduces basic concepts and structures in modern algorithms and paves the ground for more complex structures that will be discussed later. Finally, Section 2.4 provides a summary of the most significant aspects of this chapter.

2.1 Time Series Definition

A *time series* is a sequence of measurements recorded over time [9]. It can be represented as $\mathcal{X} = \{x_1, x_2, \dots, x_{T'}\}$ with length $T' \in \mathbb{N}$ and $x_t \in \mathbb{R}^c$, where c is the number of dimensions at every time step t . In a univariate time series, only one value is captured at each time step ($c = 1$), whereas a multivariate time series encompasses multiple variables that may influence one another ($c > 1$). The correlation between multiple dimensions is known as spatial or inter-metric [10].

2 Background

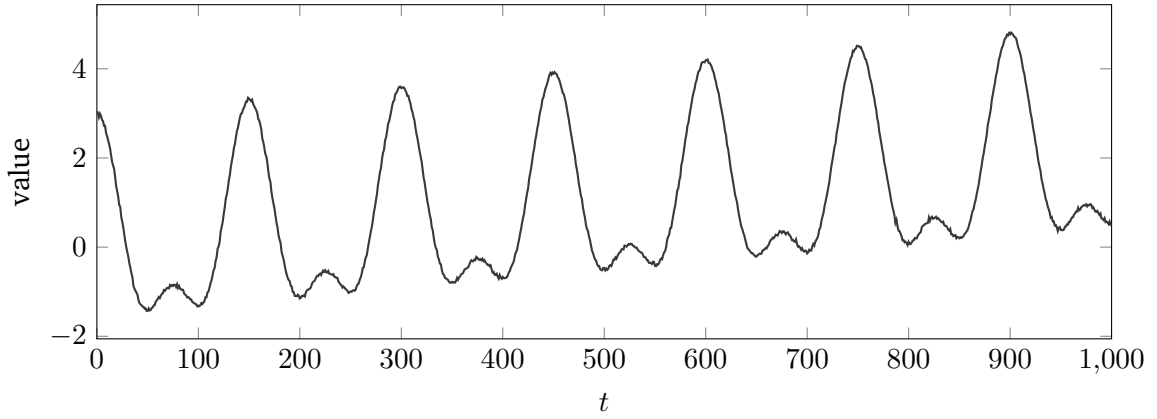


Figure 2.1: Synthetic univariate time series with seasonality and trend.

After Dodge [11], a time series can be decomposed into four descriptive components:

1. *Secular trend* is the long-term movement of the time series that can be increasing, decreasing, or stable.
2. *Seasonal variation* exists when seasonal factors influence a time series. It can be seen as a repeating pattern at a fixed frequency.
3. *Cyclical fluctuations* are also periodical variations but not as regular as seasonality and typically occur in a broader time frame.
4. *Irregular variations* are unusual events. These variations are also referred to as residual variations.

Depending on the domain, not every component has to be present. The synthetic example in Figure 2.1 shows a time series modeled after [12] as a combination of a trend function $t(\cdot)$, a function $s(\cdot)$ for seasonality and shapelet, and irregularities ϵ as

$$\mathcal{X} = s(\mathcal{T}) + t(\mathcal{T}) + \epsilon, \quad (2.1)$$

where $\mathcal{T} = 1, 2, \dots, T'$. The example shows the univariate time series similar to [13] with the basic shapelet of

$$s(t) = A \cos(2\pi\omega t) + B \sin(2\pi\omega t) \quad (2.2)$$

with ω as the frequency of the seasonality, and A and B being the amplitudes of the shapelet functions $\sin(\cdot)$ and $\cos(\cdot)$. A linear trend is chosen here as a trend function t , and ϵ is Gaussian noise with a small standard deviation.

2.2 Anomaly Detection

Anomaly detection in machine learning is a collective term for various applications. This section overviews the characteristics of anomalies in Section 2.2.1. Section 2.2.2 further

clarifies commonly used terms related to anomaly detection and emphasizes their differences.

2.2.1 Anomaly Types

From a naive point of view, the term *anomaly* describes a part of an input that does look *not normal* and is often equated with *outlier* [14, 15]. The terms outlier and outlier detection are further discussed in Section 2.2.2. This section covers the different types of anomalies, as defining them in time series is non-trivial due to temporal or spatial dependencies.

Anomalies can be categorized as *global*, *contextual*, and *collective* [6]. However, more recent studies on anomaly detection use a behavior-driven taxonomy and define *global* and *contextual* anomalies as *point anomalies* and identify three different types within the *collective anomalies*, namely *seasonal*, *trend*, and *shapelet anomalies* [13, 16]. Figure 2.2 overviews the different categories, which are discussed in detail in the following.

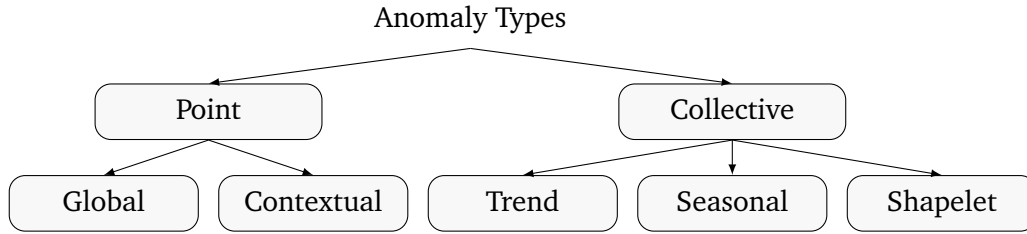


Figure 2.2: Overview of different types of anomalies.

Figure 2.3 illustrates the aforementioned types of anomalies in a synthetic univariate time series. This description and generation of anomalous time steps and patterns follow [13], which introduces a benchmark for creating synthetic anomalies. While the time series clearly displays distinct anomalies, it is important to note that multiple anomalies of different types can occur simultaneously in other scenarios.

Point Anomalies

Point anomalies affect individual time steps or at most brief intervals in the time series $\mathcal{X}$. An anomalous data point x_t at time step t with one of the two types of point anomalies can be detected by comparing its actual value with a predicted value $\hat{x}_t$ under different thresholds τ . One common criterion is

$$\|x_t - \hat{x}_t\| > \tau, \quad (2.3)$$

where $\|\cdot\|$ is a suitable norm and the prediction is the output of a model, in the simplest case, a local or global baseline $\mu_t(\mathcal{X})$.

2 Background

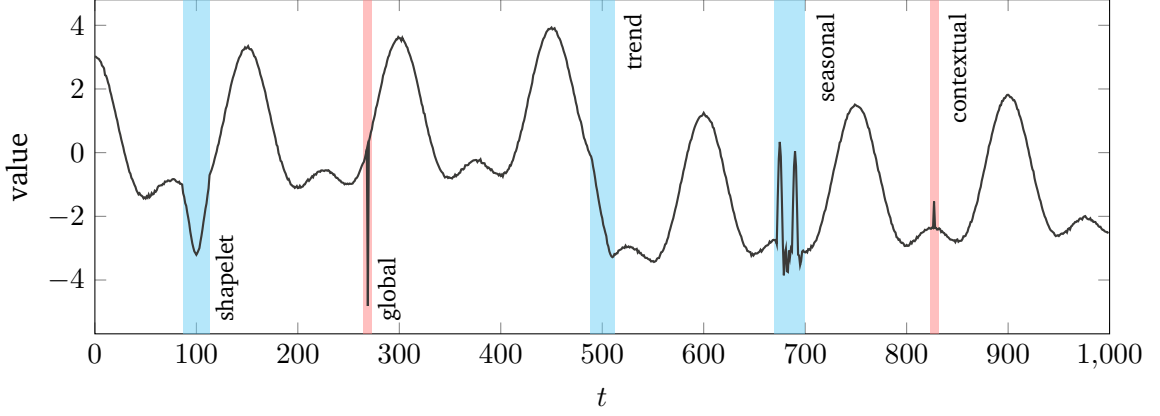


Figure 2.3: Synthetic univariate time series with different **point** and **collective** anomalies.

Global anomalies stand out upon visual inspection, as they exhibit significant deviations from the remainder of the time series. Typically, these anomalies appear as spikes when the time series has extreme values on individual time steps. At a fixed time step t , the values of $\mathbf{x}_t$ containing a global anomaly can be modeled as

$$\mathbf{x}_t = \mu_t(\mathcal{X}) + \epsilon_t, \quad (2.4)$$

where $\mu_t(\mathcal{X})$ represents the mean value of $\mathcal{X}$ at time step t and $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \lambda \sigma_t(\mathcal{X}))$ denotes a normally distributed noise term with a standard deviation that is a multiple of the standard deviation of $\mathcal{X}$, regulated by the number of standard deviations λ . Conversely, the threshold to identify these anomalies, as defined by Equation (2.3), can be determined as

$$\tau = \lambda \sigma_t(\mathcal{X}). \quad (2.5)$$

Contextual anomalies refer to a local context window around the given data point, unlike global anomalies, for which the whole time series is considered. Accordingly, Equation (2.4) can be adjusted to

$$\mathbf{x}_t = \mu_t(\mathcal{X}_{t-k,t+k}) + \epsilon_t, \quad (2.6)$$

where $\mu_t(\mathcal{X}_{t-k,t+k})$ and $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \lambda \sigma_t(\mathcal{X}_{t-k,t+k}))$ are obtained from the subsequence $\mathcal{X}_{t-k,t+k} = \{\mathbf{x}_{t-k}, \mathbf{x}_{t-k+1}, \dots, \mathbf{x}_{t+k}\}$ with k being the length to be considered before and after the current time step. Consequently, τ can be determined by

$$\tau = \lambda \sigma_t(\mathcal{X}_{t-k,t+k}). \quad (2.7)$$

Collective Anomalies

Collective anomalies affect entire subsequences $\mathcal{X}_{i,j} \subseteq \mathcal{X}$ from time step i to j and are created by different mechanisms. The detection focuses on the different components of a time series, as discussed in Section 2.1. Point anomalies can be found by comparing point-to-point values. Collective anomalies can be identified by comparing the (dis-)similarity between the components of subsequence $\mathcal{X}_{i,j}$ and the estimated one $\hat{\mathcal{X}}_{i,j}$.

Trend anomalies refer to a new trend of a subsequence compared to the remainder of the time series. In Figure 2.3, there is an increasing trend in the signal, but with an abrupt trend reversal in the middle of it. A linear trend can be seen as a low-frequency stochastic process added to the time series, as denoted in Equation (2.1). The trend in the subsequence is modeled as

$$t(\mathcal{T}_{i,j}) = -\lambda \mathcal{T}_{i,j} \quad (2.8)$$

where λ denotes the gradient of the trend. The anomaly can be found when a dissimilarity function surpasses the threshold τ

$$\text{diss}(t(\cdot), \hat{t}(\cdot)) > \tau. \quad (2.9)$$

Seasonal anomalies can be found when the subsequence shows a different seasonality than the rest of the time series. Modifying ω in Equation (2.2) results in a seasonal shift, as shown in Figure 2.3. The anomaly can be found by comparing the seasonality of different subsequences

$$\text{diss}(\omega, \hat{\omega}) > \tau, \quad (2.10)$$

where $\hat{\omega}$ is the expected seasonality.

Shapelet anomalies indicate a different underlying shapelet function for the subsequence. The anomaly can be identified by comparing the functions as

$$\text{diss}(s(\cdot), \hat{s}(\cdot)) > \tau, \quad (2.11)$$

where $\hat{s}(\cdot)$ is the shapelet function of the infected subsequence.

Finding an appropriate threshold parameter τ and defining a meaningful dissimilarity function is mandatory for identifying those anomalies. The threshold can be predefined, derived from statistical properties as shown in Equation (2.5) and (2.7), or dynamically tuned using adaptive algorithms, which are detailed in Section 5.1.

A notable algorithm for measuring the dissimilarity between two time series is Dynamic Time Warping (DTW) [17]. DTW finds the optimal alignment that minimizes the cumulative distance between the time series by allowing nonlinear temporal distortions. More specifically, the DTW algorithm creates a cost matrix $\mathbf{C} \in \mathbb{R}^{T \times T}$ such that the (i, j) -th element is the distance between the i -th element of the input time series $\mathcal{X}$ and the j -th

2 Background

element of another, e.g., its reconstruction $\hat{\mathcal{X}}$. The goal is to find a warping path W in $\mathcal{C}$ with minimal overall costs that aligns the two time series, which satisfies the underlying boundary, monotonicity, and step size constraints [17]. It can be mathematically described as

$$\text{DTW}(\mathcal{X}, \hat{\mathcal{X}}) = \min_W \sum_{i,j \in W} c(\mathbf{x}_i, \hat{\mathbf{x}}_j) , \quad (2.12)$$

where c is an arbitrary cost or distance function, e.g, Euclidean distance. While extensions of DTW exist for multivariate time series, it is most commonly applied to univariate sequences or to individual channels of multivariate data. As such, DTW does not inherently capture inter-channel dependencies, which are often critical for detecting complex anomalies. Nevertheless, DTW remains a widely used method for time series analysis [18, 19] and anomaly detection [20].

2.2.2 Detection Tasks

Based on their type and appearance, as discussed in the previous section, there are different options for anomaly detectors. An underlying procedure is to split the time series $\mathcal{X}$ into a set of smaller sequences $\mathcal{X}_{\text{train}} = \{\mathbf{X}^{(n)}\}_{n=1}^N$, $\mathbf{X}^{(n)} \in \mathbb{R}^{T \times c}$ with fixed length $T < T'$ and channels c .

Unlike anomaly types, the notions for detection tasks are not used consistently, as surveys by Darban et al. [16], Salehi et al. [21], and Yang et al. [22] show. As the number of terms closely related to anomaly detection is immense, the following list should be seen as an excerpt of the most frequently used ones.

Outlier detection is a well-studied topic in statistics and machine learning [23, 24, 25]. It concerns identifying individual observations in a dataset that are anomalous with respect to the remainder of the sample. The typical setup assumes a dataset $\mathcal{X}$ that primarily consists of nominal data. Methods either estimate a data model $p_\theta(\mathbf{X})$ for $\mathcal{X}$ or compute a robust anomaly score $S(\mathbf{X}; \mathcal{X})$. A sample $\mathbf{X}$ is flagged as an outlier if,

$$p_\theta(\mathbf{X}) < \tau \quad \text{or} \quad S(\mathbf{X}; \mathcal{X}) > \tau , \quad (2.13)$$

where the choice of p_θ , S , and the threshold τ depends on modeling assumptions. This detector can easily detect global anomalies, as defined before, when single data points have extreme values that are not present in $\mathcal{X}$.

Out-of-Distribution (OOD) detection is formulated relative to a training (in-distribution) distribution $P(\mathcal{X}_{\text{train}})$. Given an estimator or classifier trained on data from $P(\mathcal{X}_{\text{train}})$, the aim is to decide whether a new input $\mathbf{X}$ was likely generated by $P(\mathcal{X}_{\text{train}})$ or by some

other distribution [26]. OOD detection is popular in computer vision tasks [27] and more recently adapted to time series data [7].

Novelty detection assumes that only normal samples are available during the training phase. It focuses on identifying previously unseen yet related patterns during testing [28]. Formally, training provides samples from P_{normal} and novelty detection tests the hypothesis

$$H_0 : \mathbf{X} \sim P_{\text{normal}} \quad \text{vs.} \quad H_1 : \mathbf{X} \approx P_{\text{normal}} . \quad (2.14)$$

Typical approaches build a model or support region¹ for P_{normal} and flag test samples that fall outside the learned support. Novelty detection is therefore not the same as labeling any low-probability sample under a global density estimator. It is a one-class / support-estimation task with assumptions about what *normal* means.

Moreover, anomaly detection finds applications across various areas, including *fraud detection* in finance and *intrusion detection* for identifying malicious activities within networks or systems, such as cyberattacks.

As anomaly detection could be a mixture of all the mentioned techniques, it is often simplified as the identification of regions of an input that differ from or do not conform to expected normal behavior compared to other areas [6, 29] or is synonymously termed as novelty detection or outlier detection [14, 15, 21]. The term *anomaly detection* is further used as a collective term, as it is task-independent. However, with most use-cases that will be presented later on in Chapter 6 and 7, the above definition of *novelty detection* suits best.

2.3 Underlying Machine Learning Concepts

Reconstruction-based approaches are commonly used in modern machine learning architectures for anomaly detection. Models trained only on nominal data are supposed to replicate the input accurately, and notable differences in the reconstruction indicate possible anomalies. Deep learning approaches have gained popularity and outperformed conventional methods in tackling real-world problems across various industrial contexts [30]. Different traditional and modern techniques for time series anomaly detection are later discussed in Chapter 3.

The architectures covered in this chapter primarily focus on generative models and their origin. Generative Adversarial Networks (GANs), various encoder-decoder structures, and Diffusion Models were not always intended for anomaly detection or signal reconstruction, but can be successfully modified or included into different networks for these purposes.

¹A support region is an estimated region in the feature space that encloses the normal data.

2 Background

The architectures are well-known and covered in detail in several books, such as [31, 32, 33].

The following sections provide an overview of these architectures and their mathematical foundations, focusing on the most relevant approaches for this work. Given the rapidly evolving nature of deep learning research, this discussion serves as an excerpt highlighting key concepts rather than a comprehensive review. Detailed examinations and the usage of the specific architectures utilized in this thesis will be presented in subsequent chapters. The fundamental machine learning concepts are then summarized in Subsection 2.3.4 by highlighting the advantages and disadvantages of the presented models.

2.3.1 Generative Adversarial Network

GANs are a prominent class of generative deep learning networks [34]. Introduced by Goodfellow et al. [35] in 2014, a vanilla GAN can be described as a two-model approach trained in an adversarial fashion. The models and their main goal can be summarized as follows:

1. **Generator network $\mathcal{G}$:** This model learns to generate a sample corresponding to the training data distribution. It takes a random noise vector $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ as input and transforms it into a synthetic sample mimicking the real data.
2. **Discriminator network $\mathcal{D}$:** This network acts as a counterpart to the generator. It receives a batch of input samples from the training dataset $\mathcal{X}$ and from the generated samples $\mathcal{G}(\epsilon)$. Its objective is distinguishing between the two sub-batches by returning the probability of its decision $\mathcal{D}(\cdot) \in (0, 1)$. The dataset samples are often denoted as *real*, while the generated samples are denoted as *fake*.

A GAN is visualized schematically in Figure 2.4. Both models are defined as Multilayer Perceptrons (MLPs) in the original implementation [35]. However, in later publications, Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) layers are used to model spatial and temporal relationships and, therefore, extract meaningful features or generate high-quality samples [36].

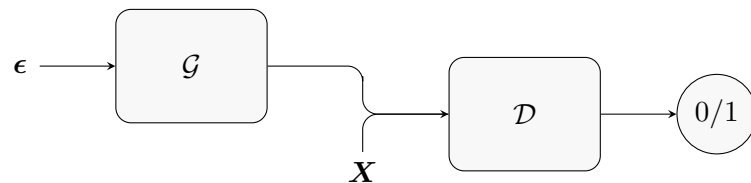


Figure 2.4: Simplified structure of a vanilla GAN

The training process of a GAN can be envisioned as a two-player minimax game with a zero-sum objective. The generator continually tries to enhance its ability to generate real-

istic samples, while the discriminator aims to improve its ability to identify fake samples [35]. The training objective of the discriminator can be summarized as the cross-entropy error function

$$\mathcal{L}_{\mathcal{D}} = -\frac{1}{N_{\text{real}}} \sum_{n \in \text{real}} \ln \mathcal{D}(\mathbf{X}^n) - \frac{1}{N_{\text{fake}}} \sum_{n \in \text{fake}} \ln (1 - \mathcal{D}(\mathcal{G}(\epsilon^n))) , \quad (2.15)$$

where the number of real samples N_{real} typically equals the number of fake samples N_{fake} . The generator is updated based on the discriminator's feedback. Its goal is to minimize the probability of the discriminator correctly classifying the generated sample as fake. Hence, the loss function of the generator can be formulated as

$$\mathcal{L}_{\mathcal{G}} = \ln (1 - \mathcal{D}(\mathcal{G}(\epsilon))) . \quad (2.16)$$

The loss of the generator relies solely on the output of the discriminator regarding the generated sample and is independent of the training data. The training loop of a GAN can be seen in Algorithm 2.1 that inherits the equations above.

Initially introduced for image generation, GANs have since evolved to cover a wide range of domains and applications. They are utilized in various computer vision tasks, including the generation of videos and three-dimensional objects, as well as in sequential data tasks, such as text and time series generation [34, 36, 37]. In addition to their primary role in generation, GANs can be applied to diverse tasks, including classification [38] and segmentation [39].

Algorithm 2.1 Training of a vanilla GAN

- 1: **repeat**
 - 2: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ sample random noise vector
 - 3: $\mathbf{X} \sim p_{\text{data}}(\mathcal{X})$ ▷ get samples from data distribution
 - 4: $\nabla_{\theta_{\mathcal{D}}} \ln \mathcal{D}(\mathbf{X}) + \ln(1 - \mathcal{D}(\mathcal{G}(\epsilon)))$ ▷ applying gradient following Eq. (2.15)
 - 5: $\nabla_{\theta_{\mathcal{G}}} \ln(1 - \mathcal{D}(\mathcal{G}(\epsilon)))$ ▷ applying gradient following Eq. (2.16)
 - 6: **until** converged
-

2.3.2 Encoder-Decoder Structures

The two-model approach of GANs can be hard to train due to known problems, like mode collapse or the failure of convergence [40] (Section 2.3.4 provides further details on the models' advantages and disadvantages.). A more intuitive strategy involves architectures, where two models or parts of a model are combined to get the desired output. Those approaches are collectively referred to as encoder-decoder structures in this thesis. This

2 Background

section primarily focuses on various architectures based on autoencoders and the Transformer model.

Autoencoder

Autoencoders have long been a cornerstone of neural networks [41, 42] with the primary objective of reconstructing the input. The latent space representation serves as the code, encapsulating all essential information required for accurate reconstruction. Formally, the goal of an autoencoder with encoder $\mathcal{E}_n$ and decoder $\mathcal{D}_e$ can be expressed as minimizing

$$\mathcal{L}(\mathbf{X}, \mathcal{D}_e(\mathcal{E}_n(\mathbf{X}))) , \quad (2.17)$$

where $\mathcal{L}$ represents the loss function, typically quantifying the reconstruction error between the input and its reconstructed version. The schematic illustration of this architecture is provided in Figure 2.5. Although simple solutions, such as directly mapping inputs to outputs without significant compression, do exist, these are impractical for real-world applications. Instead, the training process aims to induce valuable properties in the latent representation z . To achieve this, various constraints can be imposed on the model.

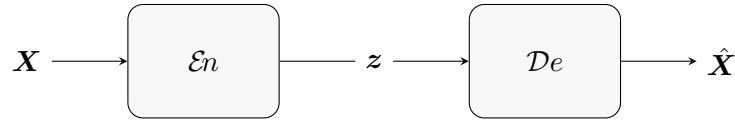


Figure 2.5: Simplified structure of a vanilla autoencoder.

One common approach involves introducing a bottleneck layer with lower dimensionality than the input. This forces the encoder to prioritize or create more abstract features of the input data. Such autoencoders are referred to as *undercomplete* and are widely employed for tasks like dimensionality reduction and feature extraction [33].²

Another strategy is to impose sparsity on the latent representation. In sparse autoencoders [43], an additional regularization term $\Omega(z)$ is incorporated into the loss function:

$$\mathcal{L}(\mathbf{X}, \mathcal{D}_e(\mathcal{E}_n(\mathbf{X}))) + \Omega(z) , \quad (2.18)$$

where Ω enforces a sparsity constraint on the latent code. This approach has proven effective to learn discriminative features in classification tasks.

²In contrast, *overcomplete* autoencoders have a code size larger than the input, which hinders the network to learn something about the data distribution.

In contrast, denoising autoencoders [44] operate by introducing noise to the input rather than modifying the loss function directly. The objective becomes to minimize

$$\mathcal{L} \left(\mathbf{X}, \mathcal{D}e \left(\mathcal{E}n \left(\tilde{\mathbf{X}} \right) \right) \right), \quad (2.19)$$

where $\tilde{\mathbf{X}}$ represents the corrupted input. By reconstructing the original data from its noisy counterpart, the model is compelled to learn robust features that capture the underlying structure of $p_{\text{data}}(\mathcal{X})$ rather than copying the input [45].

U-Net

With the evolving applications of neural networks, U-Nets were introduced by Ronneberger et al. [46] for image segmentation. Their architecture is based on an undercomplete autoencoder, which reduces input data dimensionality while retaining essential features. A key innovation is the use of skip connections, which allow information to bypass specific layers and directly connect the encoder and decoder sections. Unlike traditional autoencoders, not all information needs to be stored within a compressed representation. In the segmentation task, early features may include object edges or color values, which indicate which pixels belong to the same object. A schematic visualization of a U-Net is shown in Figure 2.6.

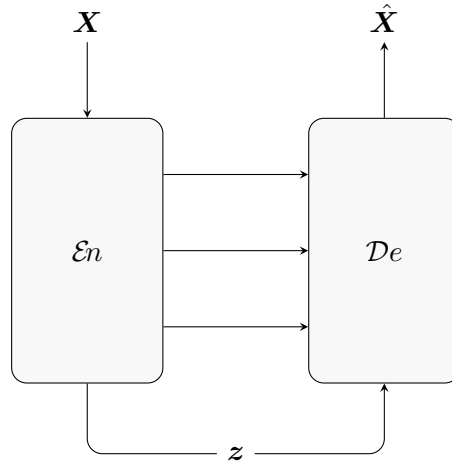


Figure 2.6: Simplified structure of a U-Net.

For semantic segmentation, in which every pixel has to be classified, the loss of the spatial information of the input would result in an inaccurate segmentation mask. The skip connections can be seen as intermediate representations of the input, helping the decoder to precisely segment the input. Recent publications have also applied this architecture to tasks besides image segmentation [47, 48].

Variational Autoencoder

While previously discussed models can be summarized as deterministic autoencoders, Variational Autoencoders (VAEs), as introduced by Kingma and Welling [49], and later by Rezende et al. [50], can be used as generative models. The overall structure, as displayed in Figure 2.7, is similar to the traditional autoencoder. Instead of directly mapping the input to a fixed latent vector, the encoder in a VAE transforms the input into the parameters of a probability distribution—typically a standard Gaussian distribution. Given the input $\mathbf{X}$, the encoder produces the mean and variance parameters, from which the latent vector $\mathbf{z}$ is sampled as

$$\mathbf{z} \sim \mathcal{N}(\mu(\mathbf{X}), \sigma^2(\mathbf{X})) . \quad (2.20)$$

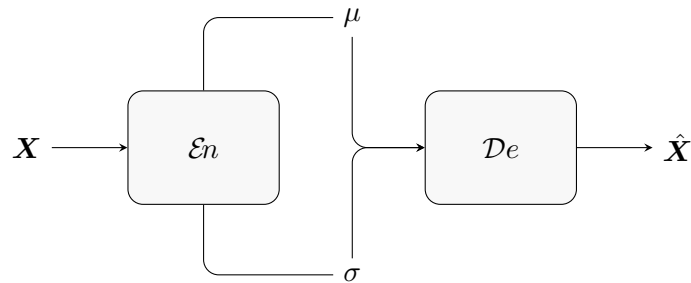


Figure 2.7: Simplified structure of a variational autoencoder.

The decoder attempts to reconstruct the input sample from a sample of the given distribution, yielding $\hat{\mathbf{X}}$. The probabilistic nature of a VAE also allows the generation of new samples by sampling from the prior distribution. The training of standard autoencoders relies solely on minimizing the reconstruction loss, as described by Equation (2.17). The generation of unseen samples involves an additional loss term that provides information on how the generated sample resembles the data in the dataset. The Evidence Lower Bound (ELBO) is thereby a crucial concept in variational inference [49], widely utilized in generative machine learning models, such as VAEs. It serves as a mechanism to approximate difficult-to-compute likelihood functions and to facilitate the optimization of probabilistic models [51]. Its significance lies in its ability to enable efficient inference and learning in complex probabilistic contexts, thereby contributing to advances in generative modeling. Training a VAE is often done by maximizing the ELBO on the data likelihood, which can be equated with minimizing

$$\mathcal{L} = \mathcal{L}_1 + \lambda \mathcal{L}_2 , \quad (2.21)$$

where $\mathcal{L}_1$ is the reconstruction loss, λ is a balancing factor, and $\mathcal{L}_2$ is a regularization term that ensures the latent representations remain close to the prior distribution. This closeness is often measured by utilizing the Kullback-Leibler (KL) divergence [52]:

$$\text{KL}(q(z) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I})) . \quad (2.22)$$

Transformer Models

Transformers represent one of the most crucial developments in deep learning. Initially introduced by Vaswani et al. [53] for Natural Language Processing (NLP), they outperform formerly used methods based on Recurrent Neural Networks (RNNs). Due to their success, Transformers were adapted to other domains, such as image or audio processing, often surpassing CNNs. A CNN can understand local relationships but struggles with long-range dependencies, as its receptive field is limited. However, those dependencies can be critical for time series data, when data is recorded at high frequency, and repetitive patterns might only be visible during long observation.

A transformer model typically consists of data preprocessing steps, multiple transformer layers, and a detection head. During preprocessing, the input is embedded into a set of vectors $\{\mathbf{x}_n\} \in \mathbb{R}^c$, where $n = 1, \dots, N$, and c represents the feature size. In the context of transformer architectures, those vectors are referred to as *tokens* that can be combined into the data matrix $\mathbf{X}$ of size $n \times c$.

The attention mechanism, initially introduced in [54] to improve RNNs, is a key building block of transformer layers. The input tokens $\{\mathbf{x}_n\}$ are transformed from their embedding space to another set of tokens $\{\mathbf{y}_n\}$ in a new space with enriched information. Each output token $\mathbf{y}_n$ is a weighted linear combination of the input vectors in the form of

$$\mathbf{y}_n = \sum_{m=1}^N a_{nm} \mathbf{x}_m , \quad (2.23)$$

where the weighting coefficients $a_{nm} = \text{SoftMax}(\mathbf{x}_n^\top \mathbf{x}_m)$ measures the normalized similarity of two input tokens. Because of its nature, the attention mechanism is also called *dot-product self-attention*.

The transformation from $\mathbf{x}$ to $\mathbf{y}$ as in Equation (2.23), re-written in a batch notation as

$$\mathbf{Y} = \text{SoftMax}(\mathbf{X} \mathbf{X}^\top) \mathbf{X} , \quad (2.24)$$

2 Background

has no learnable parameters, and all the features of the input are weighted the same. In this context, Vaswani et al. introduced *query*, *key*, and *value* matrices, denoted as $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$, which are defined by their independent linear transformation

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^q, \quad (2.25)$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}^k, \quad (2.26)$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}^v, \quad (2.27)$$

where the weight matrices $\mathbf{W}$ represent learnable parameters. In practice, the transformation is typically modeled by a single linear layer.

By replacing the input vectors with query, key, and value matrices, the attention mechanism described in Equation (2.24) can be expressed in the following matrix notation to obtain the corresponding output:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{SoftMax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_{\text{model}}}}\right) \mathbf{V}, \quad (2.28)$$

where the dot product is scaled by the key dimensionality d_{model} to prevent the softmax function from generating small gradients. In practice, a single calculation of the attention function might cause the model to average over multiple relevant patterns, resulting in an inaccurate representation. To address this, $\mathbf{X}$ is projected h times with learned linear projections, yielding separate query, key, and value matrices of dimension $d_k = d_v = d_{\text{model}}/h$ for each head. The attention as defined in Equation 2.28 is calculated h times independently and is thus called Multi-Head Attention (MHA), visualized in Figure 2.8a. The figure represents the attention module where $\mathbf{X}$ is the input. In the transformer decoder, the inputs of the attention module may vary, as the output of the encoder is used to derive $\mathbf{K}$ and $\mathbf{V}$. The outputs are concatenated and transformed, yielding the final values of $\mathbf{Y}$, and can be written as

$$\text{head}_i = \text{Attention}(\mathbf{X}\mathbf{W}_i^q, \mathbf{X}\mathbf{W}_i^k, \mathbf{X}\mathbf{W}_i^v), \quad (2.29)$$

$$\mathbf{Y} = \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{concat}(\text{head}_1, \dots, \text{head}_h) \mathbf{W}^o \quad (2.30)$$

To improve and stabilize the training of transformer layers, residual connections and normalization layers are incorporated alongside the attention blocks. The output vectors remain constrained to the subspace spanned by the input vectors. Typically, an additional MLP with a non-linearity is used to enhance the flexibility of the transformer layer.

The stackable building blocks described above form the encoder of a transformer model. The decoder is designed similarly to the encoder, utilizing the representations derived by the encoder as input. The decoder generates output sequences autoregressively, using

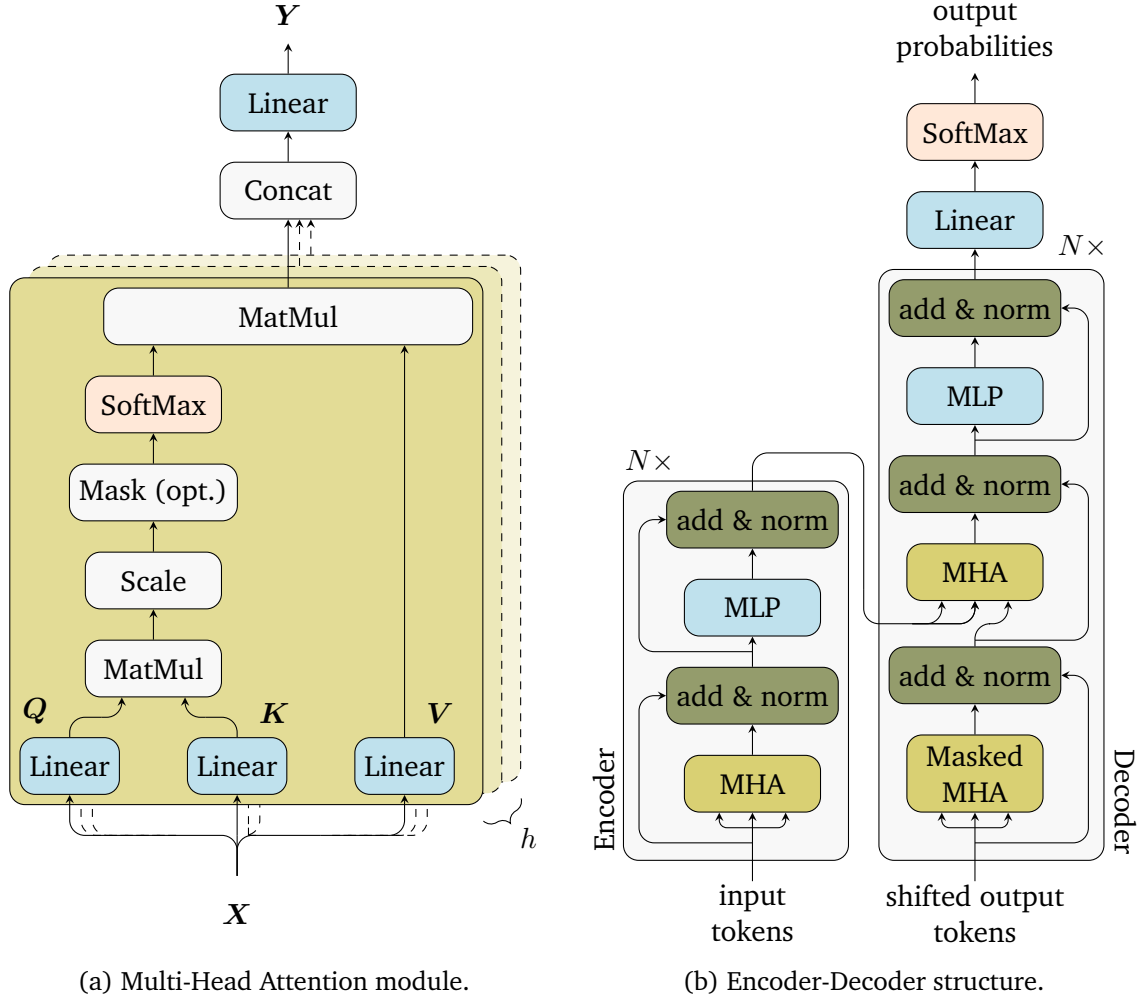


Figure 2.8: Structure of a vanilla Transformer.

previously generated tokens as additional input. The complete structure of a Transformer is displayed in Figure 2.8b.

Besides standalone Transformer networks employed for various tasks such as text or image generation [53, 55], the encoder and decoder can be used independently in diverse architectures [56, 57]. Thanks to the advantages of global input mapping, Transformers and the attention mechanism in particular are increasingly replacing convolutions in various existing architectures.

2.3.3 Diffusion Models

Another class of generative models, next to GANs, VAEs, and transformer models, is known as diffusion model or Denoising Diffusion Probabilistic Model (DDPM) [48, 58]. In contrast to previously investigated models, the neural network and its constellation of layers

2 Background

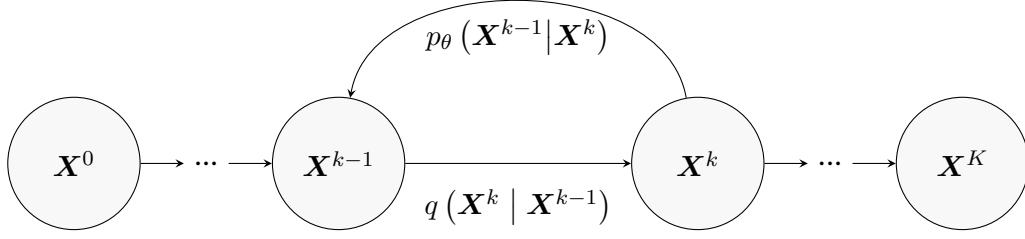


Figure 2.9: Schematic visualization of forward and backward process of a diffusion model.

are not the primary focus of diffusion models. The two cornerstones are the forward (diffusion) and the backward (denoising) process, visualized in Figure 2.9.

In the forward process, a sample $\mathbf{X}^0 \sim q(\mathbf{X}^0)$ is corrupted by gradually adding noise $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ to approximate a pure Gaussian at the final diffusion step K . This process can be described as the Markov chain

$$q(\mathbf{X}^{1:K} | \mathbf{X}^0) = \prod_{k=1}^K q(\mathbf{X}^k | \mathbf{X}^{k-1}) \quad (2.31)$$

with

$$q(\mathbf{X}^k | \mathbf{X}^{k-1}) = \mathcal{N}(\mathbf{X}^k; \sqrt{1 - \beta_k} \mathbf{X}^{k-1}, \beta_k \mathbf{I}) \quad (2.32)$$

with $\beta_k \in (0, 1)$ being the noise variance at diffusion step $k \in [1, K]$. The values of β are typically chosen by a prescribed schedule such that $\beta_1 \leq \beta_2 \leq \dots \leq \beta_K$. Each sample at diffusion step k is given by

$$\mathbf{X}^k = \sqrt{1 - \beta_k} \mathbf{X}^{k-1} + \sqrt{\beta_k} \epsilon_k. \quad (2.33)$$

It is possible to sample at any arbitrary step k in a closed loop form with $\bar{\alpha}_k := \prod_{i=1}^k \alpha_i$, $\alpha_k := 1 - \beta_k$ and $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ as

$$\mathbf{X}^k = \sqrt{\bar{\alpha}_k} \mathbf{X}^0 + \sqrt{1 - \bar{\alpha}_k} \epsilon. \quad (2.34)$$

Every irreducible positive recurrent Markov process has a backward process, which is also a Markov process. This backward denoising process is approximated by $\mathbf{X}^K \sim q(\mathbf{X}^K) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and can be described as

$$p_\theta(\mathbf{X}^{k-1} | \mathbf{X}^k) = \mathcal{N}(\mathbf{X}^{k-1}; \mu_\theta(\mathbf{X}^k, k), \Sigma_\theta(\mathbf{X}^k, k)), \quad (2.35)$$

where $\Sigma_\theta(\mathbf{X}^k, k)$ is set to $\beta_k \mathbf{I}$ and

$$\mu_\theta(\mathbf{X}^k, k) = \frac{1}{\sqrt{\alpha_k}} \left(\mathbf{X}^k - \frac{1 - \alpha_k}{\sqrt{1 - \bar{\alpha}_k}} \epsilon_\theta(\mathbf{X}^k, k) \right), \quad (2.36)$$

where ϵ_θ is a trainable function, parameterized by θ . It predicts the noise ϵ of $\mathbf{X}^k$ that is added during the forward process in Equation (2.34). ϵ_θ can be trained by minimizing the ELBO via the simplified objective function [48]

$$\mathcal{L} = \left\| \epsilon - \epsilon_\theta(\mathbf{X}^k, k) \right\|^2. \quad (2.37)$$

The primary constraint of ϵ_θ is to output a sample of the same shape as the input. The chosen model for this task was initially a U-Net, as discussed in Section 2.3.2. Predicting the added noise ϵ showed better performance in the past than directly predicting the denoised sample $\mathbf{X}^0$ [48]. The standard implementation of the training process is shown in Algorithm 2.2, where k is randomly chosen at every iteration, and the input sample is corrupted with noise accordingly. Once the model is trained to predict noise at every step k , it can generate samples from the input distribution, starting with Gaussian noise. As described in Algorithm 2.3, the sampling process is iterative, where the model is called K times to obtain the final output.

Diffusion models can be seen as a hierarchical version of VAEs, where the encoder is fixed to the forward process [59]. A one-step denoising process represents the decoder of the VAE. In

Algorithm 2.2 Training of a vanilla diffusion model

- 1: **repeat**
 - 2: $\mathbf{X}^0 \sim q(\mathbf{X}^0)$ ▷ sample input data
 - 3: $k \sim \mathcal{U}(1, K)$ ▷ sample random noise time step
 - 4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ sample random noise vector
 - 5: $\nabla_\theta \left\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_k} \mathbf{X}^0 + \sqrt{1 - \bar{\alpha}_k} \epsilon, k) \right\|^2$ ▷ applying gradient following Eq. (2.37)
 - 6: **until** converged
-

Algorithm 2.3 Generating a sample with a vanilla diffusion model

- 1: $\mathbf{X}^K \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ sample noised data
 - 2: **for** $k = K, \dots, 1$ **do** ▷ iterate through denoising steps
 - 3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $k > 1$, else $\mathbf{z} = \mathbf{0}$
 - 4: $\mathbf{X}^{k-1} = \frac{1}{\sqrt{\alpha_k}} \left(\mathbf{X}^k - \frac{1 - \alpha_k}{\sqrt{1 - \bar{\alpha}_k}} \epsilon_\theta(\mathbf{X}^k, k) \right) + \beta_k \mathbf{z}$ ▷ sample of previous noise step
 - 5: **return** $\mathbf{X}^0$
-

2.3.4 Benefits and Drawbacks of the discussed Vanilla Architectures

All the presented architectures have their respective advantages and disadvantages. The two sub-networks of a GAN can serve different purposes: A trained generator can be

2 Background

employed to generate realistic samples, which can be useful for enriching datasets [60] or producing scenarios for software-in-the-loop systems [61]. Conversely, a discriminator can be utilized to identify samples that do not conform to the training dataset. However, GANs come with well-documented challenges. They are vulnerable to mode collapse, where the generator produces similar or identical samples for varied inputs, successfully fooling the discriminator [40]. A common issue during training is that the loss of the discriminator converges to zero rapidly, resulting in a lack of gradient updates for the generator [62]. An unbalanced training can also lead to instability during training, causing the loss curves of both networks to fail to converge [63]. Later studies tried to solve these problems by continuously injecting Gaussian noise into the generator layers [64] or by modifying the loss function [65, 66].

Autoencoders and their vanilla adoptions are straightforward to use and can be customized flexibly for various types of input data. Their ability to handle complex data structures [67] made them a popular unsupervised learning method. However, a notable drawback of vanilla autoencoders is their tendency to overfit when the capacity of the latent space exceeds the input size [33] or to yield blurry reconstructions of the input [68]. The introduction of VAEs has expanded the generative capabilities of autoencoders, though their probabilistic nature introduces additional training challenges and shares limitations similar to those encountered with GANs.

Transformers are the preferred autoregressive models, primarily due to their capability to capture dependencies across the entire input through the self-attention mechanism. This mechanism allows each token to attend to all other tokens in the sequence simultaneously, enabling the efficient capture of long-range dependencies without the constraints of fixed window sizes or recurrence [53]. However, a notable drawback is the computational complexity associated with the attention mechanism. Since each token is compared with every other token, the $QK^\top$ matrix in Equation (2.28) expands quadratically with the input size, which impacts its scalability. Additionally, a limiting factor is capturing fine-grained spatial details, as all spatial locations are treated equally [69].

Diffusion models outperform previous generative techniques and are considered the SOTA in image and video generation [70, 71]. They are remarkable for their ability to generate high-resolution samples while keeping training time low. A disadvantage of using a vanilla diffusion model is the inference time, as the sample is gradually refined during generation (see Algorithm 2.3).

As every architecture is originally designed to solve a domain-specific problem, several adaptations to different data structures have emerged over the past years. Hybrid models and computational tricks are introduced to compensate for model-related disadvantages to achieve better results. Some of these models and techniques are discussed later in Section 3.2 and Chapter 4.

2.4 Summary

This chapter covered the fundamentals of this thesis. It defined the components of time series data and clarified the terms *anomaly* and *anomaly detection*. It provides mathematical definitions of various time series features to enable the generation of synthetic anomalous time series and the implementation of basic detection methods.

The introduced deep learning concepts are not explicitly designed for time series data, but they lay the groundwork for subsequent chapters and can be seen as a theoretical basis. All techniques can be assigned to the family of unsupervised models, for which no labels are needed for each sample or time step during training. This is often the case in well-known anomaly detection datasets (Chapter 6) and real-world scenarios (Chapter 7). In the following chapter, traditional and modern methods for anomaly detection and working with time series data will be covered.

3

TRADITIONAL AND CURRENT METHODS FOR ANOMALY DETECTION

Working with time series data and automatically detecting conspicuous sequences is a well-known and widely researched problem, which has been analyzed in several studies [6, 16, 72, 73]. Over the years, numerous attempts have been made to work with time series data efficiently and to identify anomalies more precisely. However, new findings and developments in the deep learning community spur scientists to achieve better results in anomaly detection metrics, close research gaps, and find solutions to well-known problems. Despite these efforts, the anomaly detection task on time series data remains incomplete.

This chapter will cover approaches for this task. Firstly, Section 3.1 addresses traditional methods for anomaly detection in time series data. The section provides an overview of various strategies for identifying anomalies, but does not explain all the methods in detail. Newer methods use deep learning-based models to extract valuable features and perform several tasks, including anomaly detection. Since this thesis primarily focuses on

generative networks, the different approaches discussed in Section 2.3 will be explored in more detail in Section 3.2. Illustrations may be based on the respective publications.

3.1 Early Works

Traditional time series analysis is grounded on various methods, which have been discussed for decades. Many of these techniques are not primarily for an anomaly detection task, but can be modified to identify unusual time points and sequences. This section outlines different categories of time series analysis in the context of anomaly detection along with examples. It is essential to note that categorization into specific categories may not always be straightforward, as it often depends on the area of application and intended use.

Statistics-based approaches. Statistics-based approaches model the data using statistical assumptions and flag points that deviate significantly from the expected value. A statistical method to detect global point anomalies easily is to normalize the data using the z-normalization [74, 24]

$$\mathbf{x}_c \leftarrow \frac{\mathbf{x}_c - \mu(\mathbf{x}_c)}{\sigma(\mathbf{x}_c)}, \quad (3.1)$$

where μ and σ represent the mean and standard deviation of the historical stationary time series $\mathbf{x}_c$ of channel c . The normalized time series is zero-centered, and point anomalies, characterized by extreme values in upcoming time steps, will be more obvious, ideally above or below a given threshold. μ and σ are estimated from the data representing the nominal state. Despite its benefits, a z-normalization cannot be applied to a non-stationary time series, due to changing statistics for different inputs and unknown statistics for future values [75]. However, the normalization can be made within a sliding window of the input time series.

Distance-based approaches. Distance-based approaches identify anomalies based on their distance from other data points. A point is considered to be an anomaly when a specified distance to a certain number of neighboring points is exceeded. For instance, when using k -Nearest Neighbors (kNN) for anomaly detection, points with a high average distance to their k nearest neighbors are classified as anomalies. Another example of a distance-based approach is DTW, as discussed in Section 2.2.1, where the distance between two time series segments is calculated. Segments that are significantly apart from each other are considered anomalies.

Clustering-based approaches. These methods partition the data into clusters and treat isolated points or small clusters as anomalies. Unlike distance-based approaches that evaluate the distance between data points directly, clustering-based methods use distance metrics to determine the similarity between data points and cluster centers. One popular method is k-means [76], where the data is split into k clusters. Initially, the cluster cen-

troids are randomly assigned, and data points are allocated to the cluster with the nearest centroid. The positions of the centroids and the assignments of the data points are then adjusted iteratively until no further changes occur. Data points that do not belong to a cluster or are significantly distant from any cluster centroid are classified as anomalies.

Density-based approaches. These methods estimate the concentration of points within the feature space, where sparse regions indicate anomalies. The first step involves transforming the data into the feature space through an embedding. To later identify sparse regions, density-based approaches also incorporate clustering as a part of their algorithm. However, they view clusters as areas of high point density that are separated by regions of low density and do not require a prior clustering step. Instead, these methods directly assess how *dense* the neighborhood surrounding each point is, flagging points located in areas of low density as anomalies [77]. A well-known example of a density-based approach is Local Outlier Factor (LOF) [78], which compares the local density of a point to the local density of its k nearest neighbors. Another approach, Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [79], clusters data points into dense regions, while points that do not belong to any cluster are identified as anomalies.

Forecasting-based approaches. Forecasting methods are used to predict future time steps based on historical data [80]. These methods can identify anomalous data points by comparing predicted values with observed ones. A widely utilized forecasting technique is the Autoregressive Integrated Moving Average (ARIMA) model [81]. ARIMA incorporates various elements of a time series, as described in Section 2.1. The autoregressive component uses the dependencies between an observation and its previous time steps. The integrated component addresses the secular trend and seasonal variations to ensure that the time series becomes stationary, which is a requirement of most statistical methods. The moving average component models the dependency between an observation and a residual error from a moving average model applied to lagged observations. By analyzing the different components, the ARIMA model can predict a limited number of future time steps. However, it is primarily suited for univariate time series data. ARIMA can also be classified as a statistically based approach [6, 74] because it relies on rigorous statistical assumptions, such as error distributions. Forecasting errors play a crucial role in identifying anomalies. Another forecasting method is the exponential smoothing model, which was introduced by Holt-Winters [82]. This approach generates forecasts based on the sum of weighted averages of past time steps, with weights decaying exponentially as the observations become older [80]. This design makes more recent information more influential in the predictions.

Reconstruction-based approaches. These approaches aim to compress the data into a latent representation and then reconstruct it. The reconstruction error serves as an indicator of potential anomalies present in the data. Introduced by Pearson in 1901, Principle

Component Analysis (PCA) [83] allows datasets to be simplified and visualized by approximating a large number of statistical variables with a smaller set of linear combinations that retain meaningful information. In an anomaly detection setup, PCA maps the input data into a low-dimensional space, as a signal with multiple dimensions can often be represented by only a few principal components without significant loss of essential information. It then reconstructs the sample using a transformation matrix, obtained from the underlying Singular Value Decomposition (SVD). A high reconstruction error suggests the presence of anomalous samples or time steps, as nominal data points should exhibit small residuals when reconstructed from the principal components [6].

Decomposition-based approaches. These methods are foundational concepts in time series analysis, as they involve breaking down the time series into its components and identifying unusual deviations. The components typically include seasonality, trend, and residuals, similar to those described in Section 2.1. Approaches based on this concept can be categorized as Seasonal-Trend Decomposition using LOESS (STL)-based methods. Although these techniques are rooted in fundamental principles, they continue to receive attention in recent publications, demonstrating satisfactory results [84, 85]. In addition to STL-based methods, time series can also be decomposed into various frequency bands using wavelet transformations, allowing for analysis at multiple resolutions [86].

Signal processing-based approaches. These methods utilize tools from signal analysis to decompose or transform the time series before performing anomaly detection. By converting samples into the frequency domain, it becomes easier to identify certain characteristics or suspicious patterns compared to analyzing the signal in the time domain [74, 6]. Anomalies may appear as unexpected spikes or drops in specific frequency bands or as significant changes in the overall energy distribution. The resulting spectrograms can be treated as images, which opens up possibilities for computer vision algorithms.

Despite their utility, traditional methods often struggle to capture long-term dependencies or spatial relationships inherent in real-world data [87]. Today, these methods have been mainly replaced by machine learning techniques. However, they still play a role as additional steps, such as preprocessing or feature generation, to simplify the training process and enhance overall performance.

3.2 Generative Models

By overcoming the computational burden of computer systems, Graphics Processing Units (GPUs) enable the identification of patterns within data with deep networks. Additionally, the introduction of user-friendly libraries such as *TensorFlow* [88] and *PyTorch* [89] has facilitated the application of machine learning to a wide range of problems.

RNNs have often been utilized in time series analysis because they can store temporal dependencies in their *memory*. The introduction of LSTM networks in 1997 [90] is often marked as a starting point for tackling time series tasks, which was later expanded with the development of Gated Recurrent Units (GRUs) in 2014 [91]. For instance, in the area of anomaly detection, Su et al. [92] combined the VAE structure with GRUs to effectively reconstruct time series data.

In recent years, generative models have become broadly used, showing superior performance in various domains and tasks compared to traditional and simple machine learning-based approaches. This section highlights recent advancements in anomaly detection using generative networks. The models discussed in the following are not restricted to the time series domain, as architectural modifications can be applied regardless of data dimensionality. Architectures based on GANs are examined in Section 3.2.1, transformer-based models are reviewed in Section 3.2.2, and diffusion process-based models are explored in Section 3.2.3. Each section includes illustrations of the various architectures, highlighting the models' compositions within the network. For more detailed, layer-level visualizations, we refer to the associated references. Our developed structures, which are discussed in Chapter 4, align with the following algorithms and are partially derived from them.

3.2.1 GAN-based Models

The algorithms discussed in this section are based on the adversarial training approach of GANs, as detailed in Section 2.3.1. A vanilla GAN consists of a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$ that are trained in a *minmax* game. This section presents different modifications of the standard architecture. Figure 3.1 provides a visual overview of various architectures.

The authors of *AnoGAN* [93] trained a convolutional GAN on medical images using the conventional approach, as illustrated in Figure 2.4. The generator $\mathcal{G}$ aims to generate samples from the data distribution based on the input vector z drawn from the latent space $\mathcal{Z}$. Once training is completed, *AnoGAN* benefits from the smooth transition within the latent space [94]: Input vectors that are closely positioned in the latent space lead to similarly generated samples. This characteristic facilitates the search for a z such that the generated sample closely resembles the input sample $\mathbf{X}$ by iteratively minimizing the residual loss function

$$\mathcal{L}_R = \|\mathbf{X} - \mathcal{G}(z)\|_1 \quad (3.2)$$

for different z as a measure of similarity. Additionally, the modified discriminator loss is given by

$$\mathcal{L}_D = \|f(\mathbf{X}) - f(\mathcal{G}(z))\|_1, \quad (3.3)$$

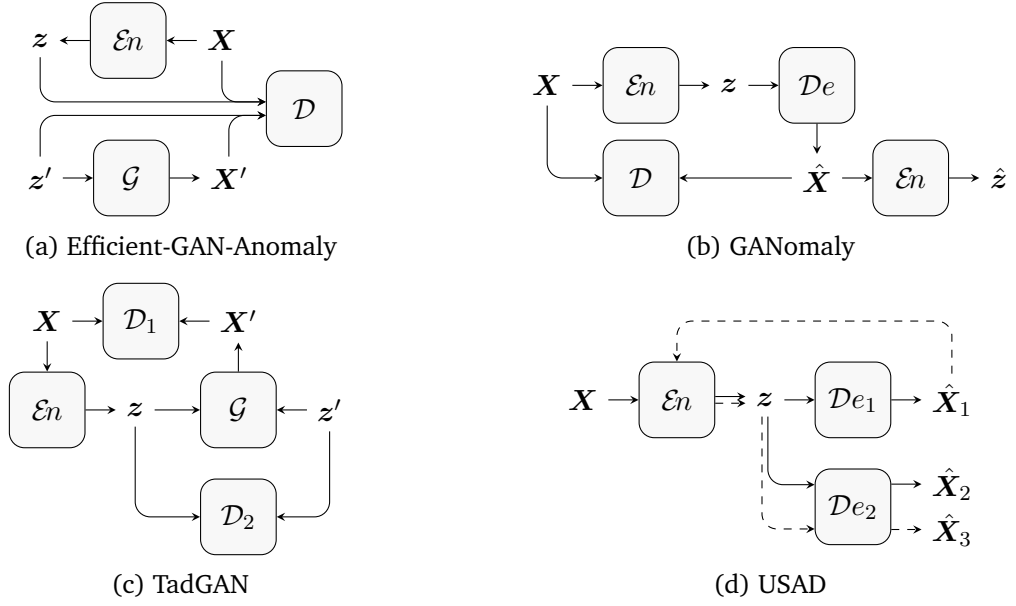


Figure 3.1: Simplified architectures of different GAN-based models.

where $f(\cdot)$ represents the output of an intermediate layer of the discriminator, ensuring that $\mathcal{G}(z)$ is consistent with the training data distribution. During this process, only the latent space coefficients of z are adjusted while keeping the weights of both the discriminator and generator fixed. A weighted combination of Equations (3.2) and (3.3) serves as the overall loss function to modify the latent space. This function is also utilized to compute the anomaly score in order to detect anomalous regions. Anomalous samples yield higher scores, as no appropriate z can be found that generates a comparable sample.

Zenati et al. replaced the step of adapting the latent space later in *Efficient-GAN-Anomaly* [95] by implementing a bidirectional GAN architecture [96] as illustrated in Figure 3.1a. This approach allows the generator to perform an inverse mapping from data to the latent representation. In addition to distinguishing between real and generated samples, the discriminator can now differentiate between a sampled latent vector and one that has been encoded. While the additional functionality requires the training of an encoder $\mathcal{E}_n$, it significantly reduces inference time by eliminating the need for iterative searches.

In *GANomaly* [97], Akcay et al. leverage the encoder $\mathcal{E}_n$ to transform the input X into its latent representation z , and use its reconstruction as the output of the generator, as depicted in Figure 3.1b. Here, the generator operates as an undercomplete autoencoder as discussed in Section 2.3.2. The output of the generator, denoted as $\hat{X}$, serves as a reconstruction of the original sample. Furthermore, a second remapping into the latent

space by the same encoder enables the calculation of an additional loss term, next to the reconstruction loss. This latent loss is defined as

$$\mathcal{L}_l = \left\| \mathcal{E}_n(\mathbf{X}) - \mathcal{E}_n(\hat{\mathbf{X}}) \right\|_2. \quad (3.4)$$

Anomaly detection is performed sample-wise using the latent loss function for each input as an anomaly score.

The authors of *MAD-GAN* [98] adhered to the original generator-discriminator architecture of a GAN but used LSTM layers instead of fully connected or convolutional layers for the purpose of time series generation and discrimination. They utilize a randomly sampled time series of the same size as $\mathbf{X}$ as a latent input. An anomaly score is computed for each time step by combining the discriminator score and the reconstruction loss of the test sample and a generated sample, where the latent variable is extracted from the input, similar to the approach used in AnoGAN [93].

Another GAN-based model for time series anomaly detection introduced by Geiger et al. [20] represents a combination of traditional GANs and the AnoGAN framework. *TadGAN*, as depicted in Figure 3.1c, employs an encoder to get a latent variable $z = \mathcal{E}_n(\mathbf{X})$ while also sampling a random z' . While the generator aims to reconstruct the original input from z , it simultaneously generates a new sample $\mathbf{X}' = \mathcal{G}(z')$. Two discriminators, $\mathcal{D}_1$ and $\mathcal{D}_2$, are trained to differentiate between original and generated time series, as well as between the encoded input and sampled latent vectors. To ensure stable and convergent adversarial training, the authors incorporated the Wasserstein distance into the loss function [65]. An anomaly score is determined by combining point- and area-wise reconstruction errors, similarity measures (DTW), and the output of discrimination $\mathcal{D}_1$.

Audibert et al. introduced a different approach to adversarial learning in *USAD* [99], which employs a GANs-inspired training of autoencoders for time series anomaly detection. Their architecture consists of two distinct decoders, $\mathcal{D}_{e1}$ and $\mathcal{D}_{e2}$, paired with a single encoder $\mathcal{E}_n$, as visually represented in Figure 3.1d. This configuration allows the model to generate different reconstructions of the same input sample, expressed as follows:

$$\hat{\mathbf{X}}_1 = \mathcal{D}_{e1}(\mathcal{E}_n(\mathbf{X})) , \quad (3.5)$$

$$\hat{\mathbf{X}}_2 = \mathcal{D}_{e2}(\mathcal{E}_n(\mathbf{X})) , \quad (3.6)$$

$$\hat{\mathbf{X}}_3 = \mathcal{D}_{e2}(\mathcal{E}_n(\hat{\mathbf{X}}_1)) . \quad (3.7)$$

The authors implement a two-phase training procedure, focused on minimizing the reconstruction loss defined in Equations (3.5) and (3.6). Additionally, as an adversarial component, the model is trained to maximize the reconstruction discrepancy of Equation (3.7). This training ensures that $\mathcal{D}_{e2}$ learns to differentiate between the latent representation of the original sample $\mathbf{X}$ and that of the already reconstructed sample $\hat{\mathbf{X}}_1$. The

anomaly score is determined as a weighted sum of the reconstruction errors with $\hat{X}_1$ and $\hat{X}_3$ relative to the input X .

Table 3.1: Overview of presented GAN-based algorithms and their particular modifications.

Model	Domain	Particularities
AnoGAN [93]	image	Inverse mapping of X by iterative search
Efficient-GAN-Anomaly [95]	Image	Inverse mapping of X with $\mathcal{E}_n$
GANomaly [97]	Image	Autoencoder as $\mathcal{G}$, latent space loss
MAD-GAN [98]	Time series	LSTM-based networks
TadGAN [20]	Time series	Autoencoder as $\mathcal{G}$, two different $\mathcal{D}$
USAD [99]	Time series	Adversarial learning of autoencoders

The architectures presented are grounded in the original adversarial learning strategy of GANs [35] and are designed specifically for anomaly detection purposes. They leverage the principle that anomalous test samples originate from a different distribution and thus cannot be effectively mapped to the latent space [20, 93, 95, 97, 98] using various techniques. Although GANs are primarily generative models, they are employed in [20, 97, 99] to reconstruct input samples as a means to identify anomalous segments at a point level. To mitigate the issue of mode collapse during training as discussed in Section 2.3.4, an additional reconstruction loss is introduced to the training process. Table 3.1 provides a summary of the discussed architectures and emphasizes their distinctions from conventional GANs.

3.2.2 Transformer-based Models

Following the insight that transformers can surpass convolutional networks in computer vision tasks with the introduction of the VisionTransformer [56], an adapted variant of the original transformer model [53], transformer-based models have been applied across various domains, including time series analysis. This section examines the key concepts behind prominent architectures that utilize transformers and their attention mechanisms.

Figure 3.2 provides an overview of several architectures, including the ordinary transformer in Figure 3.2a. Only the main transformer modules are displayed, which can be computed repeatedly in succession, as is typical in the standard process. Most models incorporate an initial embedding layer, positional encoding, and a subsequent projection layer, which are not visualized here. Unless specified otherwise in the upcoming subsections, $\mathcal{E}_n$ and $\mathcal{D}_e$ denote the transformer encoder and decoder as introduced in the relevant part of Section 2.3.2. While this work primarily emphasizes anomaly detection models, it also examines architectures for time series forecasting as done in Section 3.1, given their shared domain and data structure. In the context of time series forecasting,

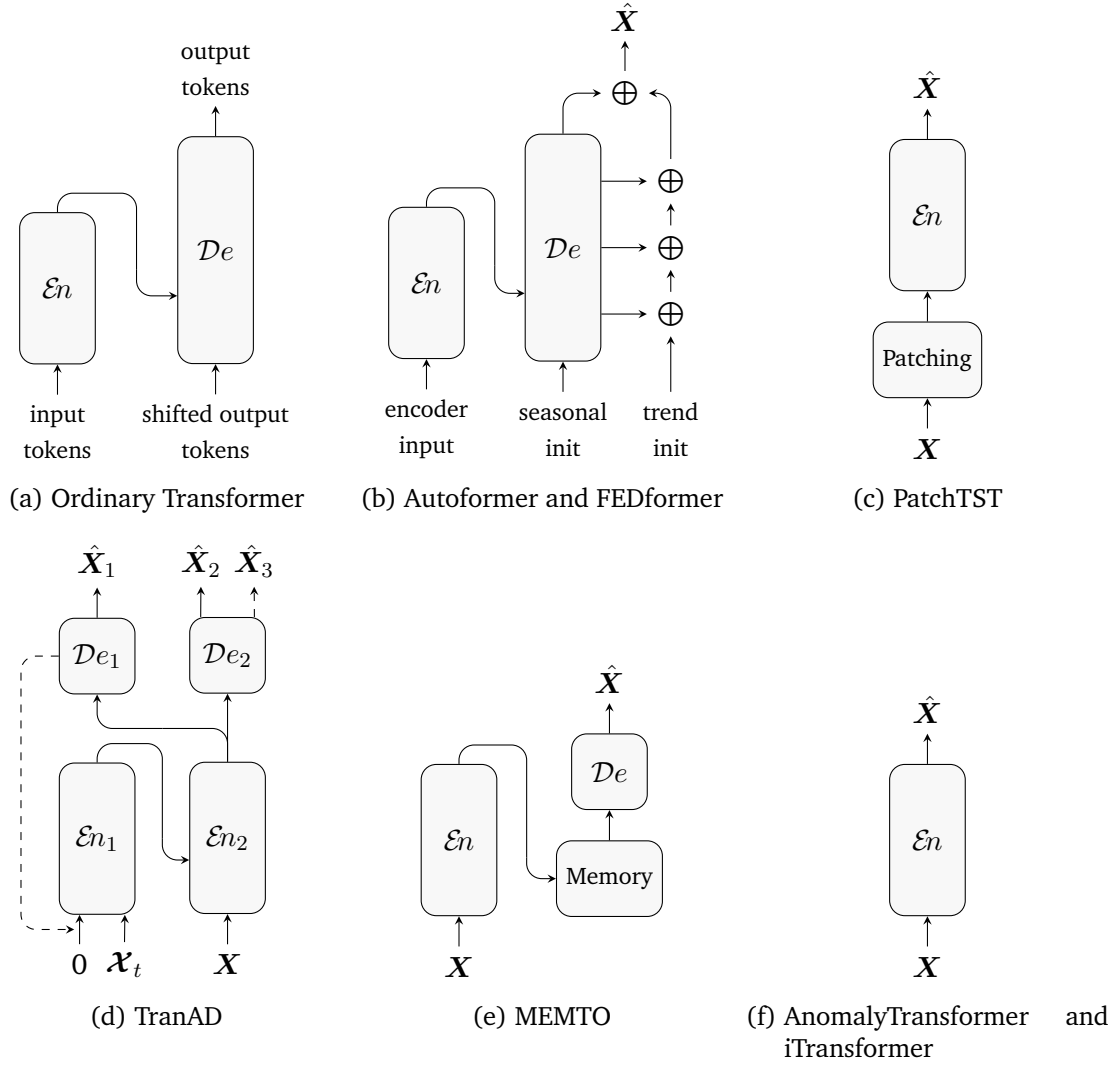
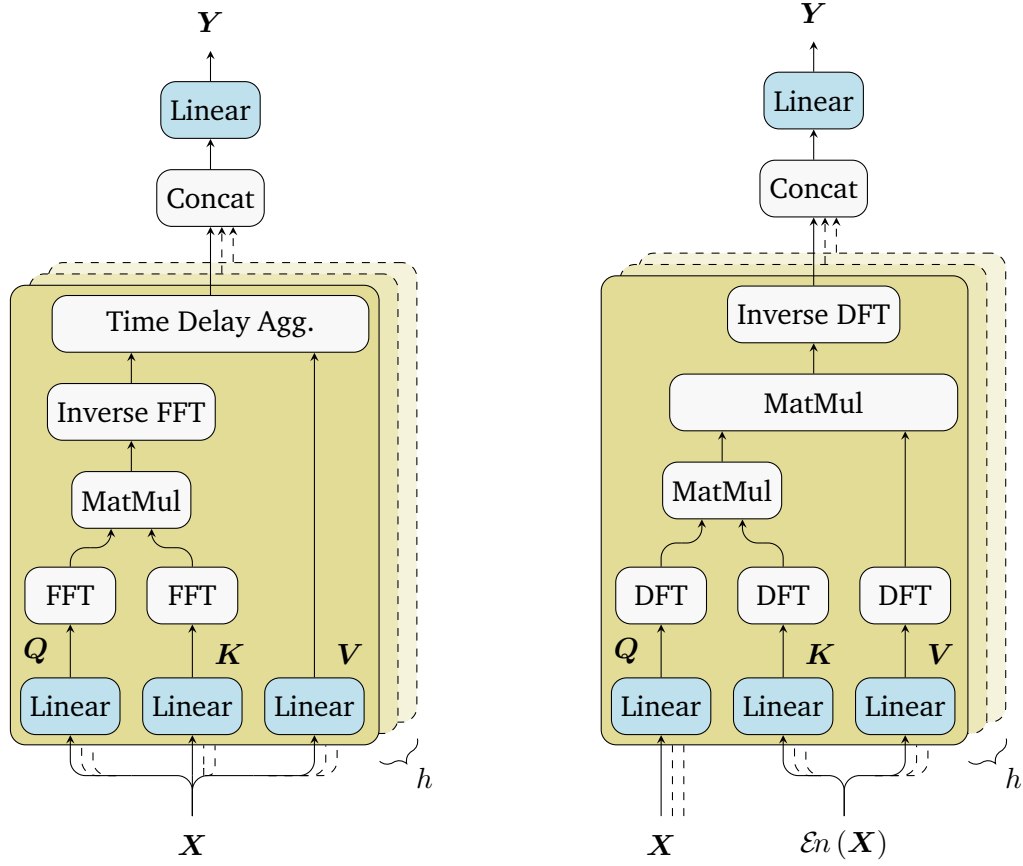


Figure 3.2: Simplified structures of different transformer-based architectures.

the objective is not to reconstruct a time series, but to predict future time steps based on historical values.

The *Autoformer* [100] and *FEDformer* [101] models preserve the transformer encoder-decoder architecture, as illustrated in Figure 3.2b, and have been designed for time series forecasting. Both models are based on the idea of time series decomposition, where the input data is first partitioned into its components (cf. Section 2.1). The trend component, which is initialized as the mean of the data, functions as an additive term for the resulting time series. Additional time series decomposition blocks are incorporated within the transformer framework alongside the attention mechanism, allowing for the removal of simpler components, such as trend and seasonality, from the input prior to modeling dependencies. These components are subsequently re-added to the overall trend. The



(a) Autocorrelation module of Autoformer.

(b) Frequency-Enhanced attention module of FEDformer.

Figure 3.3: Different extensions or adaptations of the standard attention mechanism.

primary distinction between the two networks lies in their attention calculation methodologies. The Autoformer determines the autocorrelation of the input series through Fast Fourier Transformation (FFT) to identify the top- k most significant periodic dependencies in the time domain. It employs a *Time Delay Aggregation* module to appropriately shift the series and average the representations for each of the selected lags. The attention mechanism of Autoformer is depicted in Figure 3.3a¹. In this case, attention is calculated in both the encoder and decoder, as is typical for a standard transformer. Conversely, the authors of FEDformer opted to compute attention solely in the decoder, aiming to capture global frequency patterns when the output of the encoder is used. Their frequency-enhanced attention module operates within the frequency domain, concentrating on a sparse set of Fourier or wavelet bases. Figure 3.3b showcases the attention module with features generated using Discrete Fourier Transformation (DFT).

¹Like in Figure 2.8a, the inputs are set to the ones of the first attention module. In the transformer decoder, K and V are derived from $\mathcal{E}_n(X)$.

An encoder-only approach for time series forecasting, known as *PatchTST*, was introduced by Nie et al. [102]. In this method, the multivariate time series is divided into its different channels and segmented into equal-sized smaller patches, which are then fed into an ordinary transformer encoder. The concept behind patching is to make the computationally intensive $QK^\top$ multiplication more manageable, as attention is calculated individually within each patch. However, a notable drawback of this approach is the loss of the global context of the input time series. The architecture of PatchTST is illustrated in Figure 3.2c.

The authors of *TranAD* [103] proposed an autoregressive method for reconstructing time series data. Their architecture resembles that of a conventional transformer, as depicted in Figure 2.8, and is utilized for anomaly detection through the standard attention mechanism as outlined in Equation (2.28). Analogous to USAD [99], they implemented an adversarial two-phase training strategy with two Encoders, $\mathcal{E}_{n_1}$ and $\mathcal{E}_{n_2}$, as well as two Decoders, $\mathcal{D}_{e_1}$ and $\mathcal{D}_{e_2}$. While $\mathcal{E}_{n_1}$ functions as the standard transformer encoder, $\mathcal{E}_{n_2}$ represents the initial part of a transformer decoder leading up to the MHA module. The decoder is comprised of a MLP followed by a sigmoid function. In the first phase, the model generates an approximate reconstruction of the input window. The second phase is conditioned on the subsequences, where the reconstruction is inaccurate. The model can be summarized as

$$z_1 = \mathcal{E}_{n_2}(\mathcal{E}_{n_1}([\mathcal{X}_t, 0]), \mathbf{X}) , \quad (3.8)$$

$$\hat{\mathbf{X}}_1 = \mathcal{D}_{e_1}(z_1) , \quad (3.9)$$

$$\hat{\mathbf{X}}_2 = \mathcal{D}_{e_2}(z_1) , \quad (3.10)$$

$$z_2 = \mathcal{E}_{n_2}\left(\mathcal{E}_{n_1}\left([\mathcal{X}_t, \|\hat{\mathbf{X}}_1 - \mathbf{X}\|_2]\right), \mathbf{X}\right) , \quad (3.11)$$

$$\hat{\mathbf{X}}_3 = \mathcal{D}_{e_2}(z_2) , \quad (3.12)$$

with $\mathcal{X}_t$ being the complete input time series up to the current time step t . As in USAD, all the reconstructions $\hat{\mathbf{X}}_n$ are used for loss calculations during training and to calculate an anomaly score α during testing as

$$\alpha(\mathbf{X}) = \frac{1}{2}\|\hat{\mathbf{X}}_1 - \mathbf{X}\|_2 + \frac{1}{2}\|\hat{\mathbf{X}}_3 - \mathbf{X}\|_2 . \quad (3.13)$$

MEMTO [104] is a reconstruction-based model designed for time series anomaly detection. As illustrated in Figure 3.2e, its fundamental architecture comprises a transformer encoder and a *weak* decoder, which consists of two fully connected layers. The encoder transforms the input into a latent representation, while the decoder, being less powerful than the encoder, cannot achieve an accurate reconstruction. This limitation allows the network to yield significantly poorer reconstruction results when the input includes an

anomalous segment. Additionally, the authors introduced a memory module within the latent space to prevent the model from over-generalization and to increase the difficulty of the reconstruction task, particularly in the presence of anomalies.

Xu et al. presented an alternative approach known as *AnomalyTransformer* [105]. The authors employed the encoder of the traditional transformer model to reconstruct an input sample, as illustrated in Figure 3.2f. Their proposed *anomaly attention* module is founded on association discrepancy, enhancing the self-attention through an additional learnable transformation of the input as visualized in Figure 3.4. This transformation yields the scale parameter $\sigma = \mathbf{X}\mathbf{W}^\sigma$, which is used to compute the prior association $\mathcal{P}$ for each attention head² as

$$\mathcal{P} = \text{Rescale} \left([G(|j - i|; \sigma_i)]_{i,j \in \{1, \dots, T\}} \right) \quad (3.14)$$

with the Gaussian kernel defined by

$$G(|j - i|; \sigma_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp \left(-\frac{|j - i|^2}{2\sigma_i^2} \right), \quad (3.15)$$

which calculates the association weight from the i -th point to the j -th point. Their proposed series-association $\mathcal{S}$ corresponds to the attention matrix of the standard transformer

$$\mathcal{S} = \text{SoftMax} \left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \right), \quad (3.16)$$

with $\mathbf{Q}$ and $\mathbf{K}$ being the linear transformations of the input as with the ordinary attention mechanism. Their encoder-only approach uses L encoder layers. The association discrepancy AssDis at layer l is characterized as a symmetrized KL divergence between $\mathcal{P}$ and $\mathcal{S}$ for each attention head and is part of the training objective alongside the reconstruction error. The association discrepancy can be written as

$$\text{AssDis}(\mathcal{P}, \mathcal{S}; \mathbf{X}) = \left[\frac{1}{L} \sum_{l=1}^L \left(\text{KL}(\mathcal{P}_{t,:}^l \parallel \mathcal{S}_{t,:}^l) + \text{KL}(\mathcal{S}_{t,:}^l \parallel \mathcal{P}_{t,:}^l) \right) \right]_{t=1, \dots, T} \quad (3.17)$$

The authors of *iTransformer* [106] proposed inverting the time series to treat each signal channel as a separate token for their encoder-only transformer. The overall architecture of this model is equal to that of *AnomalyTransformer* as depicted in Figure 3.2f, but used for time series forecasting. By inverting the time series, the attention matrix as an output of the $\mathbf{Q}\mathbf{K}^\top$ -calculation at each layer can be interpreted as a correlation matrix between different variates of the input time series, thereby capturing multivariate correlations.

As summarized in Table 3.2, the transformer architectures discussed can be divided into two main categories: forecasting and anomaly detection approaches. Each algorithm

²For clarity and readability, the superscript designators for the attention heads are omitted.

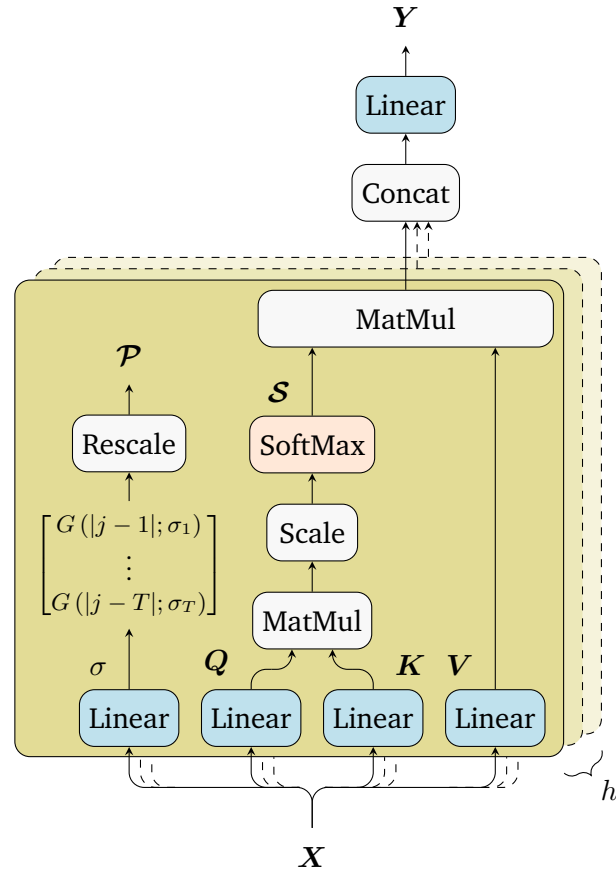


Figure 3.4: Attention module of AnomalyTransformer.

Table 3.2: Overview of presented transformer-based algorithms for time series data and their particular modifications.

Model	Task	Particularities
Autoformer [100]	Forecasting	Autocorrelation attention
FEDformer [101]	Forecasting	Frequency attention
PatchTST [102]	Forecasting	Patching of time series
TranAD [103]	Anomaly Detection	Adversarial Training
MEMTO [104]	Anomaly Detection	Gated Memory
AnomalyTransformer [105]	Anomaly Detection	Anomaly attention
iTransformer [106]	Forecasting	Inverted dimensions

employs a distinct approach for computing attention or model composition. Modifications of the attention module are tailored for time series-specific tasks as in [100, 101, 105]. Additionally, model-specific adjustments [103, 104] and modifications of the input data [102, 106] contribute to achieving SOTA results for anomaly detection tasks.

3.2.3 Diffusion-based Models

In the context of diffusion models, as discussed in Section 2.3.3, the architecture responsible for predicting the added noise does not necessarily take the central role. The key inquiry is how to direct the model to produce a specific output starting from a Gaussian noise vector.

In recent years, researchers have developed strategies to generate more realistic outputs and guide the model toward the desired output. For numerous tasks such as image generation, an effective approach is to introduce a conditioning or context vector c^3 as an additional input to the model that specifies the target output. The loss function for diffusion models, as in Equation (2.37), can be extended to

$$\mathcal{L} = \left\| \epsilon - \epsilon_\theta \left(X^k, k, c \right) \right\|_2^2. \quad (3.18)$$

Depending on the specific task and data domain, the conditioning could take the form of an embedded text input [107] or class labels [108]. However, these methods appear less practical for time series data, particularly in the context of anomaly detection, where the goal is to reconstruct the input signal as long as it is free of anomalies. In contrast, time series forecasting presents a more straightforward task, where the context typically involves the observed past [109] or its learned representation [110, 111].

Previous approaches to time series anomaly detection have primarily focused on input reconstruction. However, with traditional DDPMs structures, a sample of the learned dis-

³It is not a mandatory requirement for c to be in vector form, it is often handled as such.

tribution is generated. In its original form, it is therefore not suitable to reconstruct a sample when starting the process with pure noise $\mathbf{X}^K \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ without prior knowledge of the input. One strategy is to frame anomaly detection as an imputation problem, where a subsection of the input is masked, and the objective is to reconstruct only this masked segment. A mask is typically a matrix with the same shape as the input data, containing zeros and ones. An element-wise multiplication of the mask and the signal yields the masked signal, which is used as the condition. For the following approaches, a synthetic sample is constructed, following the definition in Section 2.1 and applied different conditioning techniques. For the sake of readability, only one channel ($c = 0$) is displayed.

The authors of *DiffAD* [112] aimed to identify healthy time steps within a sample, masking the remaining steps that should be imputed. A key challenge lies in identifying time steps that reflect nominal states. In *DiffAD*, a density-based strategy is utilized to select normal points. While those strategies can be used as a standalone anomaly detector as discussed in Section 3.1, the authors use them to compute coarse pre-labels on the raw time series. They compute the density ratio of two consecutive (non-overlapping) input windows across the dataset to identify potential normal and anomalous subsequences. To remove distortions in the normal subsequences, they linearly interpolate the time series values between equally spaced points of each subsequence. The values of the identified anomalous subsequences are also interpolated linearly, but without prior splitting. The re-assembled time series is multiplied by the binary pre-label and used as a condition⁴. The resulting condition is illustrated in Figure 3.5. Despite their approach to detecting anomalies on multivariate time series, they only utilize the first channel to get the mask and copy the interpolated time series to the other channels of the condition. Thus, the condition and the input time series have the same dimensions. As an internal network, they utilize a U-Net with *ResNet* blocks [113] and attention calculations (cf. Equation (2.28)).

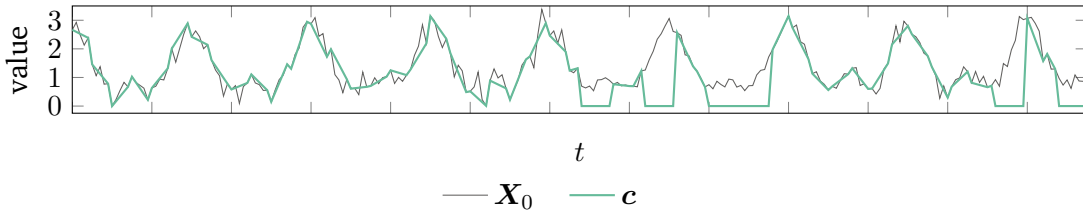


Figure 3.5: Example of conditioning channel $c = 0$ of input $\mathbf{X}$ with *DiffAD*.

In a similar manner, *ImDiffusion* [114] was proposed as an imputation-based time series anomaly detection approach. This method applies two complementary masks to an input sample, predicts independently the imputation outcomes, and merges these results to produce a sample where each time step is derived from the output of the model. The

⁴While the multiplication with the mask is clearly explained in the publication, it is omitted in the implementation, resulting in the interpolated signal as the condition.

complementary masked conditions are visualized in Figure 3.6. The authors of ImDiffusion also designed an internal model for their approach. Instead of a U-Net, they used a sequence of residual blocks, each containing two transformer layers. With the transformer layers, the global relations are calculated both temporally and spatially.

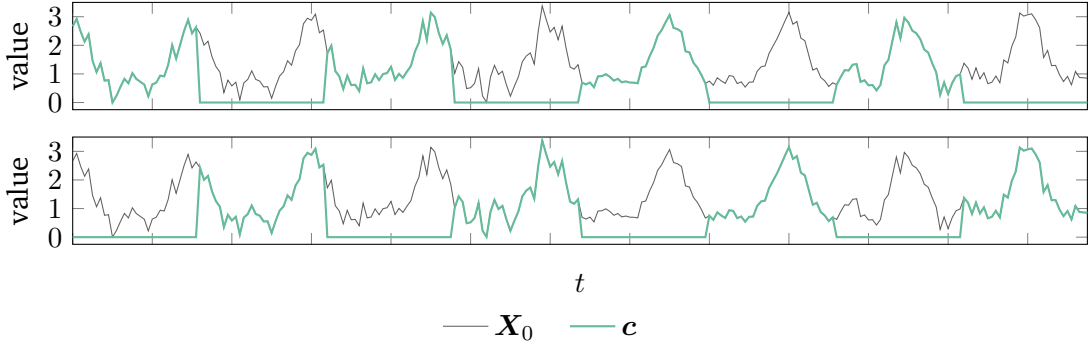


Figure 3.6: Example of conditioning channel $c = 0$ of input $\mathbf{X}$ with ImDiffusion.

An alternative approach to guiding the diffusion process in time series anomaly detection networks involves identifying meaningful input representations prior to starting the diffusion process. In *DiffusionAE* [115], the input is first reconstructed using an under-complete transformer-based autoencoder before being fed into the diffusion process. This approach indirectly conditions the diffusion on the output of the autoencoder. While this primarily affects the forward process and the calculation of noisy samples, the authors further employed the autoencoder and the forward process for signal reconstruction during inference. They start the generation process with a noisy time series $\mathbf{X}^k$ with $k < K$, which is derived from the forward process and thus retains information about the input sample, thereby eliminating the need for additional context vectors. While the autoencoder is transformer-based, the model for denoising is a conventional U-Net with *ResNet* blocks.

Likewise, *DDMT* [116] utilized a convolution-based autoencoder for signal reconstruction in the first stage. The reconstruction error is used as a metric for the masking algorithm. Time steps with significant discrepancies between the original and reconstruction are masked in the original input data, as well as their neighboring points when they are closely related to each other as visualized in Figure 3.7. The masked time series is used as a condition for the internal neural network of the denoising process. This internal network, mainly consisting of transformer encoder layers, learns to generate a sample similar to the input and imputes the missing subsequences.

In summary, diffusion-based approaches for time series anomaly detection often focus on masking regions and use them as additional context for the model [112, 114, 116] or start the denoising process with a corrupted input sample [115]. Table 3.3 overviews the

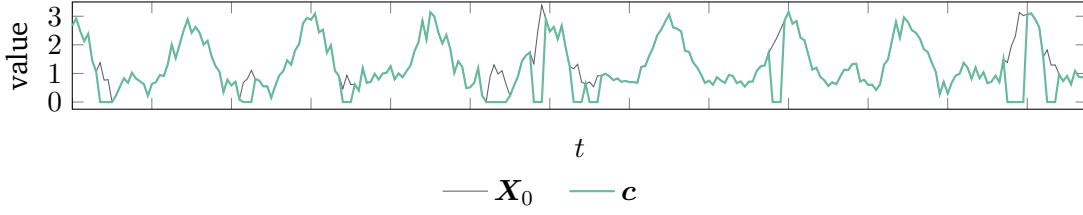


Figure 3.7: Example of conditioning channel $c = 0$ of input $\mathbf{X}$ with DDMT.

proposed algorithms and their innovations in relation to ordinary DDPMs. All models are trainable end-to-end.

Table 3.3: Overview of presented DDPM-based algorithms for time series anomaly detection and their particular modifications.

Model	Task	Particularities
DiffAD [112]	Imputation	Identification of normal time steps
ImDiffusion [114]	Imputation	Contrary masking
DiffusionAE [115]	Reconstruction	Autoencoder, sampling from $\mathbf{X}^k$
DDMT [116]	Imputation	Masking by autoencoder reconstruction error

3.3 Summary

This chapter discussed several techniques for analyzing time series data that have emerged over the past decades.

Traditional methods continue to be relevant in specific scenarios, particularly when the data is relatively simple or when computational resources are limited. In modern approaches, these traditional techniques are often integrated as additional steps or as components of a bigger model. For deep learning-based methods, preprocessing step such as dimensionality reductions [98, 117] or normalization of the time series data to a specific range or zero-mean value [20, 99, 103, 115] can be advantageous, supporting the model in learning patterns and, in conclusion, enhancing its ability to detect anomalies. It is important to note that while reducing data dimensions can be beneficial, it may also result in challenges during anomaly detection. Even if there is no information loss in the training data, a reduction in the test data can result in eliminating anomalies. Although dimensionality reduction and normalization are largely applicable across various domains, time series-specific methods, such as decompositions and signal processing techniques, are incorporated as structural components in nowadays models [100, 101].

In addition to various combinations of traditional and modern architectures, a synthesis of modern approaches can be found that aims at overcoming model-specific limitations.

While autoencoder networks may be considered outdated, they can serve as foundational elements in GAN-based architectures [20, 97, 99] or act as a prior network for diffusion models [115, 116]. Besides the pure transformer networks mentioned, the transformer (encoder) or the attention mechanism are frequently utilized as layers in different architectures [114, 116]. Due to their ability to capture long-range dependencies more efficiently than convolutional layers do [53], they are used within U-Net structures of diffusion models. In [118], the authors have integrated a GAN discriminator with a diffusion model to improve generation quality. The diffusion model can thereby be seen as the generator of a GAN. Using the conventional transformer structure as a standalone architecture for signal reconstruction can cause a reconstruction of anomalous segments, as each token relies heavily on its own and neighboring representations [116].

A persistent challenge across all the mentioned architectures is the requirement for a relatively limited input window size of the sample. While larger windows can enhance contextual understanding [100, 119], they also lead to difficulties in accurately reconstructing the input [120]. Table 3.4 outlines the algorithms specifically designed for time series anomaly detection, including their proposed input window size according to their publications or implementations. The preference for smaller input windows results from various factors: computational challenges (particularly for transformers), a shortage of publicly available long time series with high contextual dependencies (as discussed later in Chapter 6), and the choice of window-based anomaly detection, where the anomaly score is assigned to an entire input sample, such as when using the discriminator output of a vanilla GAN. The authors of MAD-GAN experimented with various input sizes without explicitly specifying the one that yields the proposed result. In their publication, the authors of DiffAD did not disclose input data related to hyperparameters like input sample sizes or window sizes for pre-labeling. However, their corresponding implementation uses an input sample size of $T = 128$ for two datasets and $T = 2,048$ for two other datasets.

Table 3.4: Input window sizes of presented time series anomaly detection algorithms.

Model	Input window size
MAD-GAN	30 – 300
TadGAN	100
USAD	10
TranAD	10
MEMTO	100
AnomalyTr.	100
DiffAD	128/2,048
DDMT	100
DiffusionAE	100
ImDiffusion	100

Another key challenge for all anomaly detection approaches is to find the right balance between generalization and overgeneralization. On one hand, models should be designed to generalize effectively, enabling them to adapt to unseen (test) data and reconstruct input accurately. On the other hand, they must avoid the ability to reconstruct anomalous parts, regardless of their appearance.

The following Chapter 4 introduces novel techniques for time series anomaly detection that build on the same foundational architectures as the approaches presented here. In Chapter 6, we further place these techniques and our proposed models in context using publicly available datasets.

4

NOVEL GENERATIVE DEEP LEARNING MODELS FOR TIME SERIES ANOMALY DETECTION

An anomaly, as defined in Section 2.2, refers to a rare event with distinct characteristics. As the majority of data points are of normal behavior, traditional classification algorithms struggle due to significant class imbalances. A common approach is to train a deep learning model that learns the typical patterns and distributions within the training data, which is assumed to represent only the nominal state. Reconstruction-based methods are often utilized to reconstruct fixed-length input samples. For example, with an encoder-decoder structure, as discussed in Section 2.3.2, the model learns to encode an input sample and then decode it from the latent representation by minimizing the resulting reconstruction errors.

During testing, the model can successfully encode and decode normal-looking samples. However, the reconstruction of anomalous samples results in one of two scenarios: either the latent representation is unfamiliar to the decoder since it was not seen during training,

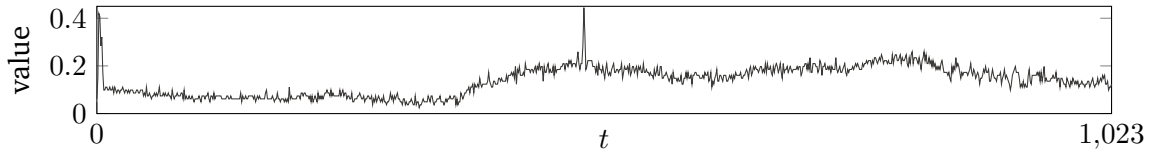


Figure 4.1: Snippet of the training data.

failing to reconstruct it; or the latent representation of the anomalous sample is similar to that of a normal sample, leading to a reconstruction of this normal sample, without anomalous subsequences. This difference in the reconstruction results in a high error when compared to the original input sample. For most models (cf. Section 3.2), the reconstruction error serves as the leading indicator for the presence of an anomaly.

This chapter presents novel techniques for deep learning-based anomaly detection in time series data, utilizing generative models that follow reconstruction-based approaches. The primary objective is to train a model $f(\cdot)$ for a training dataset $\mathcal{X}_{\text{train}} = \{\mathbf{X}^{(n)}\}_{n=1}^N$, consisting of (overlapping) samples $\mathbf{X}^{(n)} \in \mathbb{R}^{T \times c}$ of fixed length T and c channels. The reconstructed sample $\hat{\mathbf{X}}^{(n)} = f(\mathbf{X}^{(n)})$ has the same shape as the input. The superscript (n) is omitted in the following sections for readability reasons. The anomaly score $\mathbf{a}_t$ associated with each time step $t = 1, \dots, T$ of each sample is thereby not exclusively derived by the reconstruction error on a time step level but can inherit other measurements as well (e.g., similarity scores between subsequences or the output of another model). This chapter focuses primarily on novel deep learning models based on generative approaches to the anomaly detection problem. Chapter 5.2 will focus on various metrics for evaluation.

Before describing the innovative models, we introduce a time series that will be used in this chapter as an illustrative example. We focus on a single time series from Server Machine Dataset (SMD), a publicly available dataset for time series anomaly detection. This time series, entitled `machine-1-6` in the following, contains $c = 38$ channels for each of 23,688 time steps for training and testing. Test data is thereby labeled in a binary manner, indicating where an anomaly is present. This and other datasets will be introduced in Chapter 6 in more detail. Figure 4.1 illustrates a single channel of a subsection of this time series.

The sections of this chapter are structured as follows: Section 4.1 introduces the general hardware setup and implementation details for model training and evaluation. Section 4.2 introduces a GAN-based approach for time series anomaly detection, including a description of common problems when designing GAN structures and suitable options to eliminate them. Section 4.3 introduces a transformer-based approach that can be utilized for long time series contexts. Section 4.4 employs diffusion models in the context of anomaly detection, highlighting advances in the forward process. In Section ??, combinations of introduced networks are shown before summarizing this chapter in Section 4.6.

4.1 General Setup for Training and Evaluating Models

The experiments and their time and memory measurements were carried out on a computer running Ubuntu 20.04. This system is equipped with an 11th Gen Intel Core i7-11700K processor operating at 3.60GHz, featuring 16 cores, 64GB of memory, and a single Nvidia RTX 3080Ti GPU with 24GB of memory. The PyTorch library [89] with CUDA support in Python version 3.10 was used throughout the experiments.

Training machine learning models requires tracking hyperparameter configurations, training performance metrics, and results. Modern deep learning frameworks have numerous hyperparameters and configurations that require monitoring. These range from model-specific parameters, such as model choice, model depth, and layer dimensions, to dataset-specific parameters, such as input size, and training-specific parameters, including the choice of optimizer and the number of training epochs.

In the presented framework, different configurations are managed using YAML files that inherit all relevant parameters. Furthermore, training sessions with varying configurations and evaluations of other settings can be easily compared and reproduced by using the open-source library MLflow [121], which stores the metadata of each run locally. Next to training parameters, model and system metrics can be graphically monitored during the training process. Additionally, the trained model can be saved for further reuse. As a visualization tool for the user interface during inference, the Gradio library [122] was utilized.

4.2 Generative Adversarial Networks with Stabilized Training

Disclaimer: Parts of this section were already published under the title: *A Generative Model for Anomaly Detection in Time Series Data* (2022) [123] and *Generative Anomaly Detection in Multivariate Time Series* (2023) [124]. However, hyperparameters or model-specific layers may have changed.

The vanilla GAN [35] as discussed in Section 2.3.1 was introduced for an image generation task as a two-model minimax game, where a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$ are adversarially trained to surpass each other, thereby increasing overall performance. The samples from the dataset are often referred to as *real*, while generated samples are referred to as *fake*.

One major challenge of traditional GANs is that the equilibrium between the sub-networks cannot be guaranteed even when their internal design is similar in model depth and layer structure. When a strong generator produces high-quality samples, a discriminator is not able to distinguish between real and fake ones. On the other hand, a strong discriminator always classifies its input correctly, which results in a generator that does not

make progress due to vanishing gradients [35]. This appears frequently at an early stage during training, when the task of the discriminator is simple, as the generator samples are of poor quality.

The training of the generator relies solely on the discriminator's output regarding the generated samples (cf. Equation (2.16)). A known problem is that the same sample is generated repeatedly, regardless of the input vector $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, as the generator learns that the discriminator does not recognize it as fake. This problem is known as mode collapse and originates from the weights of early generator layers, which level out variations in ϵ [40].

Section 4.2.1 discusses the mentioned drawbacks and presents solutions that enhance training stability and improve results. This chapter further highlights the potential of GANs for time series anomaly detection. Several modifications were already introduced in Section 3.2.1 that utilize a GAN for this task. In Section 4.2.2, novel GAN-based algorithms are introduced.

4.2.1 Eliminating Disadvantages of GANs

A solution to support an equilibrium between $\mathcal{G}$ and $\mathcal{D}$ is to compute the gradient of the less performative network, typically the discriminator, more often in one iteration than the stronger one [35]. This is implemented as an additional internal loop during training. A suggestion to eliminate the vanishing gradient problem is to modify the loss function for the generator. Instead of minimizing Equation (2.16) the generator should maximize $\ln \mathcal{D}(\mathcal{G}(\epsilon))$ for more stable gradients [35]. However, despite their simplicity, these approaches are not always effective in enhancing the training process and do not entirely solve the problems of mode collapse or vanishing gradients.

A different approach involves modifications of the loss functions. Arjovsky et al. [65] introduced the Wasserstein distance [125] as a loss function to measure the distance between two distributions, which is proven to stabilize the training of a GAN. Instead of using a discriminator to classify or predict the probability of generated samples as real or fake, the Wasserstein GAN (WGAN) replaces the discriminator model with a critic that scores the realness or fakeness of a given input. It tries to enlarge the output for real samples and keep the output for fake samples low. The loss functions of a WGAN are written as

$$\mathcal{L}_{\mathcal{D}} = -\mathcal{D}(\mathbf{X}) + \mathcal{D}(\mathcal{G}(\epsilon)) , \quad (4.1)$$

$$\mathcal{L}_{\mathcal{G}} = -\mathcal{D}(\mathcal{G}(\epsilon)) , \quad (4.2)$$

where the discriminator, or critic, tries to maximize the difference of its output regarding real and fake inputs. Conversely, the generator tries to maximize the discriminator's output

for fake samples. In other words, the generator aims to minimize the distance between the distribution of the training data and the distribution observed in generated samples. A requirement for $\mathcal{D}$ in a WGAN is to follow 1-Lipschitz continuity, which is realized in practice by weight clipping.

The loss function for the discriminator was further extended in [66] with a soft gradient norm penalty to approximate the Lipschitz constraint to

$$\mathcal{L}_{\mathcal{D}} = \mathcal{D}(\mathbf{X}) - \mathcal{D}(\mathcal{G}(\epsilon)) + \lambda_{\text{gp}} (\|\nabla_{\bar{\mathbf{X}}} \mathcal{D}(\bar{\mathbf{X}})\|_2 - 1)^2, \quad (4.3)$$

where $\bar{\mathbf{X}}$ is sampled uniformly between real and fake samples and λ_{gp} is the gradient penalty coefficient (commonly $\lambda_{\text{gp}} = 10$). This loss function results in more stable gradients and prevents them from vanishing (or exploding) without weight clipping.

Label smoothing is another popular approach to prevent the discriminator loss from converging quickly [40]. Labels indicate whether the input sample of the discriminator is real or fake. The labels are typically set to -1 and 1 when using the WGAN loss and therefore omitted in the loss functions, such as in Equation (4.1). Incorporating small amounts of noise into the labels helps prevent the discriminator from becoming overly confident. Also, the injection of noise into the discriminator input has an impact on the certainty of the discriminator's decisions.

In an anomaly detection setup, the generator of the GAN can be utilized to reconstruct the input sample accurately. The discriminator should be able to distinguish between the original and the reconstruction. An autoencoder used as the generator, as in [97, 20], is able to produce these reconstructions and thereby eliminate mode collapse.

The following list summarizes techniques for stabilizing and strengthening a GAN. In addition to the ones mentioned before, additional adjustments are listed. According to the loss functions, both networks depend on each other's output, so adjustments to a single network also affect the other.

1. **Objective modification:** Replacing the classic binary cross-entropy loss functions can stabilize a GAN. The Wasserstein distance is often utilized because it yields smoother gradients, thereby preventing gradient vanishing and exploding.
2. **Architectural adjustments:** As the networks' depth impacts their success, designing one network weaker prevents it from converging quickly. Additional layers, such as dropouts, can be further used to complicate the training process. For the reconstruction-based approach, the generation process is conditioned on the input sample in the form of an autoencoder.
3. **Input manipulation:** The real and fake data and the assigned label are modified by inducing a small portion of Gaussian noise to prevent the discriminator from becoming overconfident.

4. **Training schedule adaptation:** Training the weaker network more frequently per training iteration adjusts the balance. As the two networks have their own optimizer, different learning rates are also utilized to adjust the gradient steps.

4.2.2 Conditional GAN for Anomaly Detection

When developing GAN-based approaches for time series anomaly detection, a range of model design choices are integrated in addition to the ones mentioned to address the limitations associated with traditional GANs, resulting in novel architectures. This section describes our approaches to regularizing these networks and how to use them to identify anomalies. Our primary focus is on the generator, as its output plays a crucial role in identifying anomalies in time series data. Modifications to the discriminator compared to its traditional use are limited in our setups. Our discriminator $\mathcal{D}$, a network consisting of ConvBlocks, produces one value for each of the smaller patches of the input, similar to [47]. This approach allows for more local decisions than the traditional discriminator, which outputs only a single value for each input.

AE-GAN

The first design choice involves the modification of z , as done by others, which is the input of the generator $\mathcal{G}$. Unlike with traditional generators that generate a sample given a random vector, the generator of an AE-GAN is conditioned on the real samples $\mathbf{X}$ as an undercomplete convolutional autoencoder and trained to produce reconstructions, similarly as introduced in [97] for image data. Our generator is visualized in Figure 4.2. The convolutional blocks (ConvBlocks) compress two one-dimensional strided convolutional layers followed by non-linearity and normalization layers for the downstream task. The TConvBlocks in the decoder use a transposed convolution to upsample the latent representation instead. The reconstruction is expressed as

$$z = \mathcal{G}_{\mathcal{E}_n}(\mathbf{X}) , \quad (4.4)$$

$$\hat{\mathbf{X}} = \mathcal{G}_{\mathcal{D}_e}(z) . \quad (4.5)$$

Encoder $\mathcal{G}_{\mathcal{E}n}$ and decoder $\mathcal{G}_{\mathcal{D}e}$ are typically symmetrically structured, containing the same number of convolutional blocks with equal-sized parameters. By utilizing the WGAN-GP loss, the loss functions for the networks in the setup are expressed as

$$\mathcal{L}_{\mathcal{D}} = \mathcal{D}(\mathbf{X}) - \mathcal{D}(\mathcal{G}(\mathbf{X})) + \lambda_{\text{gp}} (\|\nabla_{\tilde{\mathbf{X}}} \mathcal{D}(\tilde{\mathbf{X}})\|_2 - 1)^2, \quad (4.6)$$

$$\mathcal{L}_{\mathcal{G}} = \lambda_{\text{adv}} \mathcal{L}_{\text{adv}} + \lambda_{\text{rec}} \mathcal{L}_{\text{rec}}, \quad (4.7)$$

$$\mathcal{L}_{\text{adv}, \mathcal{G}} = -\mathcal{D}(\mathcal{G}(\mathbf{X})), \quad (4.8)$$

$$\mathcal{L}_{\text{rec}} = \|\mathcal{G}(\mathbf{X}) - \mathbf{X}\|_2, \quad (4.9)$$

where the loss terms for $\mathcal{G}$ are regularized by their $\lambda \geq 0$ factor and $\mathcal{L}_{\mathcal{D}}$ is adapted from Equation (4.3). We refer to this network as AE-GAN in the latter.

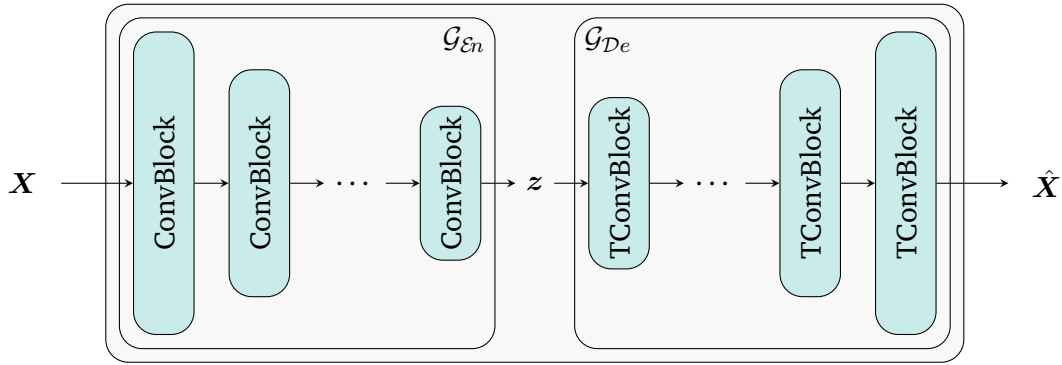


Figure 4.2: Conditional generator as an autoencoder.

An approach to further improve the generator's performance is to compare the input and its reconstruction in the latent space. With an accurate reconstruction of the input signal, the latent representations z and $\hat{z}$ should be the same. As with the reconstruction discrepancy in the time domain, a reconstruction error in the latent domain is calculated as part of the loss function for the generator. The latent representation of the reconstruction $\hat{z}$ is written as

$$\hat{z} = \mathcal{G}_{\mathcal{E}n}(\hat{\mathbf{X}}). \quad (4.10)$$

While this network structure is not a novel approach [126], it lays the foundation for our modifications and serves as a reference algorithm in the following sections and chapters.

Sparse-AE-GAN

To make the autoencoder more robust to changes in the input data [43], we leverage the discussed benefits of AE-GAN by sparsifying the latent representation z in [123] and name this approach Sparse-AE-GAN in the following. Sparse autoencoders, as discussed earlier

in Section 2.3.2, commonly enforce sparsity by penalizing the latent representation z for being dense. This penalty term is calculated as an additional loss term, e.g., as the L1 norm of the latent representation, which promotes sparsity, or the KL-divergence between the average activation of each latent neuron ρ_j and a small target ρ [127]. As an activation, the sigmoid is often used to ensure the value range of $[0, 1]$. Those loss functions are defined as

$$\mathcal{L}_{\text{sparse,L1}} = \|z\|_1, \quad (4.11)$$

$$\mathcal{L}_{\text{sparse,KL}} = \sum_{j=1}^m \text{KL}(\hat{\rho} \parallel \rho_j). \quad (4.12)$$

Zhou et al. [128] followed a different approach by employing a small Sparse Regularization Network (SRN), inspired by sparse LSTM cells [129], that takes the latent representation as an input and returns a sparse version of it, which is then used for reconstruction. The reconstruction process can be summarized as

$$\hat{X} = \mathcal{G}_{\mathcal{D}_e}(\text{SRN}(\mathcal{G}_{\mathcal{E}_n}(X))) \quad (4.13)$$

in this case. The regularization network is trained by minimizing the Mean Squared Error (MSE) between the encoder output and the sparse representation as

$$\mathcal{L}_{\text{sparse,SRN}} = \frac{1}{m} \|z - \text{SRN}(z)\|_2^2, \quad (4.14)$$

where m corresponds to the number of latent neurons.

Each loss function associated with the different versions of sparse coding encourages intrinsic sparsity in the encoder's output. Two of these loss calculations require additional hyperparameters, as summarized in Table 4.1. The overall generator loss of Equation (4.7) can be expanded for the training of a *Sparse-AE-GAN* to

$$\mathcal{L}_{\mathcal{G}} = \lambda_{\text{adv}} \mathcal{L}_{\text{adv}} + \lambda_{\text{rec}} \mathcal{L}_{\text{rec}} + \lambda_{\text{sparse}} \mathcal{L}_{\text{sparse}, \cdot}, \quad (4.15)$$

where the loss terms are regularized by their $\lambda \geq 0$ factor. The loss function for the discriminator remains as earlier. In our publication [123], we trained the network to detect anomalies in audio data, thereby having deeper convolution-based networks to process large signals. However, the overall model structure and the main idea behind its composition are adaptable to different time series domains. We will apply this network in Chapter 6 and 7 to various time series data.

Table 4.1: Additional hyperparameters for different sparsity calculations.

Method	Hyperparameters
L1	-
KL	target value ρ
SRN	network depth, hidden size

Dual-AE-GAN

While the decoder in Figure 4.2 is symmetrical to the encoder, we employ in another novel GAN-based model design a second *weak* decoder $\mathcal{G}_{De'}$ in the generator [124]. The weakness is implemented with a reduced number of blocks, thereby making the reconstruction less accurate. The kernel size for transposed convolution must be adjusted to avoid dimensionality mismatches in the output. The poor reconstruction supports the generator or the discriminator, depending on the usage. On the one hand, the poor reconstruction, treated as a fake sample, can help the discriminator training, as it eases the task of distinguishing between real and reconstructed samples. On the other hand, the model design supports the generator encoder in generating more informative latent representations, as the second decoder has to reconstruct the input with limited capabilities. Furthermore, a weak decoder can be helpful in specific scenarios, as in [104], which uses its output to enhance time series anomaly detection, as discussed in Section 3.2.2. Figure 4.3 shows the generator with its two decoders. The weak reconstruction process is written similarly to Equation (4.5) as

$$\hat{\mathbf{X}}' = \mathcal{G}_{De'}(z) . \quad (4.16)$$

The network with this extended autoencoder setup is referred to as *Dual-AE-GAN* in the latter. The weak decoder is learned alongside the other components of the generator, making the model end-to-end trainable. By having multiple trained networks, we are more flexible in combining them when calculating anomaly scores for a detection task, but at the cost of more hyperparameters to optimize and longer training sessions. In the setup introduced, the discriminators' input for *fake* samples is a batch containing a random combination of both reconstructions.

Summary and Experimental Studies

All the presented GAN-based networks are summarized in Table 4.2 and were initially implemented using ConvBlocks, as described earlier, to condense the input and preserve temporal relations. In the following experiments, four ConvBlocks for $\mathcal{G}_{En}$, $\mathcal{G}_{De}$ and $\mathcal{D}$ are used and two blocks for $\mathcal{G}_{De'}$.

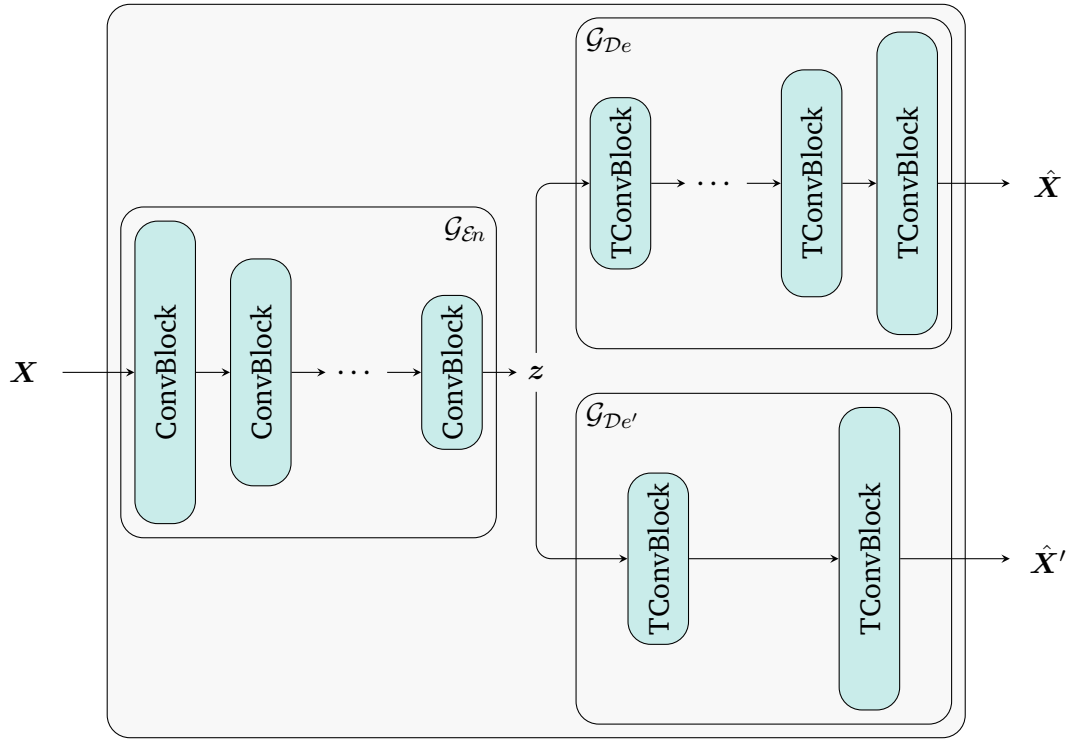


Figure 4.3: Conditional generator as an autoencoder with a second decoder.

Table 4.2: Overview of the different GAN implementations

Name	structure of $\mathcal{G}$
AE-GAN	autoencoder
Sparse-AE-GAN	autoencoder with sparse latent representation
Dual-AE-GAN	autoencoder with strong and weak decoder

Finding the equilibrium of generator and discriminator is one of the major challenges when training GANs. Different adjustments were presented and applied to the traditional GAN in this section to stabilize the training. By introducing sparsity to the latent representation of the input, reconstructing a sample becomes more challenging for the generator, and, on the other hand, the objective of the discriminator to distinguish between real and fake samples becomes easier.

Experiments with publicly available data demonstrate a rapid convergence of the discriminator's loss curve, whereas the adversarial loss of the generator remains volatile during training and does not necessarily converge. Figure 4.4 shows the resulting loss curves for $\mathcal{L}_{\mathcal{D}}$ and $\mathcal{L}_{\text{adv},\mathcal{G}}$ when using *AE-GAN* and *Sparse-AE-GAN* with the different sparsity techniques. The loss curves, especially those for the generator, illustrate the instability of

training a GAN. Each model configuration is trained for 5 runs, where varying outcomes can be observed.

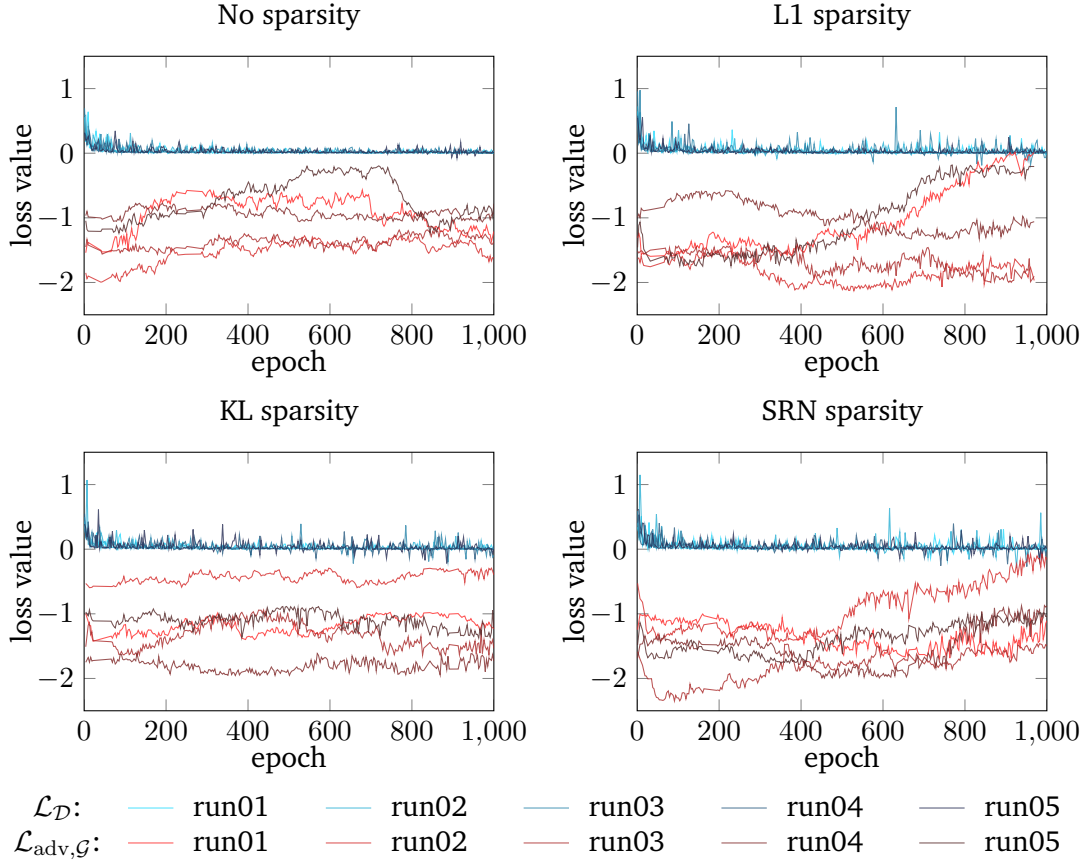


Figure 4.4: Impact of different sparsity regularization techniques on the adversarial loss of discriminator and generator.

The generator is updated three times per discriminator update to enhance its ability to reconstruct the input samples. Furthermore, the dropout attached to the discriminator is set to a higher level than the generator (0.3 vs 0.1). Label smoothing is activated, and noise is injected into the real samples in 10% of the training iterations. The discriminator loss quickly converges towards zero, even when a warm-up phase is included for the generator, which focuses on its ability to reconstruct the input. For illustrative purposes, all the models are trained for 1,000 epochs on an input window length of 64, utilizing the AdamW optimizer [130] with the standard configuration ($\beta_1 = 0.90$ and $\beta_2 = 0.99$) and a learning rate of 0.0001. For the experiments, the Wasserstein distance with gradient penalty serves as an adversarial loss function (Equation (4.6) and (4.8)). The reconstruction loss, as in Equation (4.9), and the dedicated sparse loss function complete the loss for the generator as in Equation (4.15). The sparse loss is not computed for the AE-GAN model. Each training took around 90 minutes, including setup and configuration as pro-

posed. The Sparse-AE-GAN with SRN sparsity takes slightly longer training time, as there are additional weights to train.

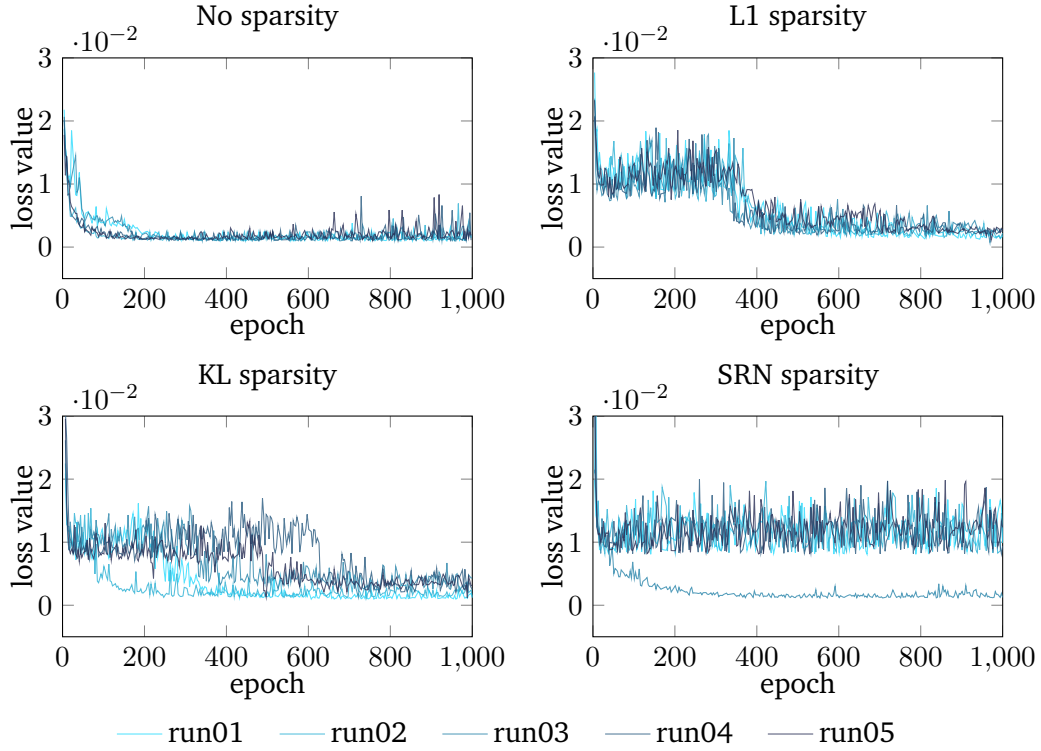


Figure 4.5: Impact of different sparsity regularization techniques on the reconstruction loss of discriminator and generator.

A closer look at the specific loss curves of the generator reveals the effect of the different sparsification techniques. Figure 4.5 visualizes the impact of sparsity on the reconstruction loss $\mathcal{L}_{\text{sparse}}$ in the training mentioned above. The sparsity weakens the reconstruction capabilities of the generator during the early stage of the training. With the additional L1 and KL loss, the reconstruction results at the end of the training are on par with the loss where the latent representation is not further penalized. The SRN technique causes the reconstruction to be on the same level, and no further improvement is observed during later training stages.

In the Dual-AE-GAN architecture, the encoder is forced to encode the most valuable features into the latent representation, as both decoders, the strong and the weak one, try to reconstruct the sample from the same z . Both reconstruction losses form the overall reconstruction loss of the generator as a convex combination. When utilizing the experimental setup and configurations as before, a more stable training process is observed compared to AE-GAN and the sparse versions. The stability is evident when examining the loss curves directly. While the combined loss for the discriminator quickly converges to zero, the adversarial loss of the generator first oscillates around zero before converging.

A further indicator of training stability is the reconstruction losses, which are without single spikes, as seen with the other discussed sparse GAN-based approaches. These loss curves are displayed in Figure 4.6, which also reveal the slightly faster convergence when reconstructing the input with the decoder of the same size as the encoder. The training of the Dual-AE-GAN model takes 110 minutes.

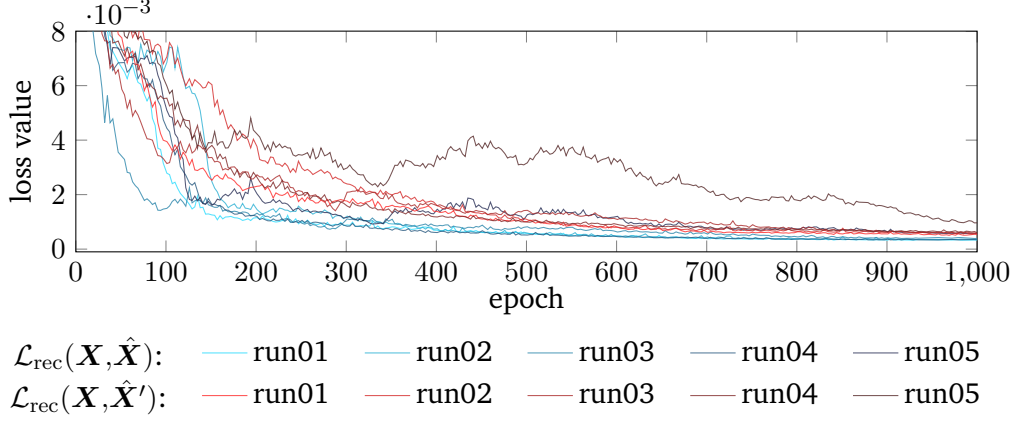


Figure 4.6: Reconstruction loss curves of different runs with the Dual-AE-GAN setup.

4.3 Scalable Transformer Model for Large Input Windows

The classical Transformer architecture [53], as displayed earlier in Figure 2.8, has been adapted for the time series domain and anomaly detection tasks (see Section 3.2.2) in various versions. Inspired by recent advancements in transformer architectures, this study utilizes these models to reconstruct multivariate time series and detect anomalies. Time series data can be complex, depending on the sensors and sampling frequencies used, which makes a flexible approach according to the input shape beneficial. While datasets typically have a fixed number of channels c , which can be reduced with dimension reduction techniques, such as PCA, they remain flexible regarding sequence length T .

However, a central problem of vanilla transformer architectures is the limited input size, as the complexity rises quadratically with the input length. We evaluate different strategies to reduce the complexity of the attention calculation in Section 4.3.1. Section 2.3 described several possibilities to reconstruct an input signal in order to perform anomaly detection. Section 4.3.2 introduces a novel technique based on well-known encoder-decoder structures, in which transformer layers are inherited. Furthermore, Section 4.3.3 shows experimental results with different hyperparameter settings and identifies preferable configurations.

4.3.1 Improving Attention Calculation to Reduce Complexity

Numerous publications have explored new attention mechanisms to reduce the complexity of the original attention mechanism and to embed domain-specific features. This section compares several of those techniques in terms of complexity and their applicability to novel model structures and includes pseudocode for a better understanding. The primary focus is on the projection of $\mathbf{X} \rightarrow \mathbf{Y}$ with $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{T \times c}$ through the Multi-Head Attention (MHA) mechanism, as previously discussed and illustrated earlier (see Section 2.3.2 and Figure 2.8a).

Standard Attention [53]. The ordinary transformer model marked the introduction of the attention mechanism in a deep learning context, capturing all pairwise dependencies within the input. It is expressed in a simplified form, as previously shown in a one attention head manner, as

$$\mathbf{Y} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{SoftMax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right) \mathbf{V}, \quad (4.17)$$

where the quadratic complexity lies within the $\mathbf{Q}\mathbf{K}^\top$ multiplication of the complete transformed input represented by $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$. In Algorithm 4.1, this transformation is performed as a fused projection, which is a common procedure in practice. Additionally, the algorithm illustrates the MHA, where attention is calculated individually for H heads. The resulting attention matrices are combined with a final projection layer to yield the output.

Algorithm 4.1 Standard Attention

Require: $\mathbf{X} \in \mathbb{R}^{T \times c}$, heads H

Ensure: $\mathbf{Y} \in \mathbb{R}^{T \times c}$

- 1: $[\mathbf{Q}, \mathbf{K}, \mathbf{V}] \leftarrow \mathbf{X}\mathbf{W}^{qkv}$ ▷ fused QKV projection
 - 2: reshape $\mathbf{Q}, \mathbf{K}, \mathbf{V} \rightarrow [H \times T \times d]$ ▷ $d = c/H$
 - 3: **for** each head $h = 1, \dots, H$ **do**
 - 4: $\mathbf{A}_h \leftarrow \text{SoftMax}(\mathbf{Q}_h \mathbf{K}_h^\top / \sqrt{d}) \mathbf{V}_h$
 - 5: $\mathbf{A} \leftarrow \text{reshape}(\text{concat}(\mathbf{A}_1, \dots, \mathbf{A}_H))$
 - 6: **return** $\mathbf{Y} = \mathbf{A}\mathbf{W}^o$
-

Monotonic Chunkwise Attention [131]. This approach segments the sequence into fixed-length windows (chunks) of size w . The standard attention is computed within each window separately before reassembling the output. Pseudocode for the foundational architecture is presented in Algorithm 4.2, where the windows are not overlapping, resulting in no cross-chunk interactions.

Swin Attention [132]. The authors of the Shifted window (Swin) Transformer present a mechanism, originally in a hierarchical order for object detection in images, where the

Algorithm 4.2 Monotonic Chunkwise Attention

Require: $\mathbf{X} \in \mathbb{R}^{T \times c}$, chunk size w , heads H
Ensure: $\mathbf{Y} \in \mathbb{R}^{T \times c}$

- 1: Partition $\mathbf{X}$ into $\lceil T/w \rceil$ chunks $\{\mathbf{X}^{(i)}\}$ of length w
- 2: **for** each chunk $\mathbf{X}^{(i)}$ **do**
- 3: $\mathbf{A}^{(i)} \leftarrow \text{StandardAttention}(\mathbf{X}^{(i)}, H)$
- 4: Reassemble $\{\mathbf{A}^{(i)}\}$ into $\mathbf{A} \in \mathbb{R}^{T \times c}$
- 5: **return** $\mathbf{Y} = \mathbf{A} \mathbf{W}^o$

input is partitioned into smaller, non-overlapping windows of size w and computed by the standard attention mechanism¹ in a manner similar to [131]. In practice, Algorithm 4.3 is utilized to calculate the attention in a constellation with two subsequent transformer encoders. The first encoder computes attention within the windows without partitioning shifts ($s = 0$). Conversely, in the second encoder, the local windows are shifted by a value of $0 < s < w$, typically $s = w/2$. This shift results in a new composition of sub-windows, allowing cross-window connections and thereby enhancing the information flow throughout the entire sequence. In a subsequent publication [133], the authors introduced a scaled cosine attention mechanism to replace the standard attention in the Swin Transformer, which yields softer attention values through the normalization of the cosine function.

Algorithm 4.3 Swin Attention

Require: $\mathbf{X} \in \mathbb{R}^{T \times c}$, window size w , shift size s , heads H
Ensure: $\mathbf{Y} \in \mathbb{R}^{T \times c}$

- 1: **function** PARTITION($\mathbf{X}, \delta$)
- 2: $\mathbf{X}' \leftarrow \text{roll}(\mathbf{X}, -\delta)$; pad $\mathbf{X}'$ to multiple of w
- 3: **return** reshape $\mathbf{X}'$ into windows of length w
- 4: $\mathcal{W} \leftarrow \text{Partition}(\mathbf{X}, s)$
- 5: **for** each window $\mathcal{W}^{(i)}$ **do**
- 6: $\mathbf{A}^{(i)} \leftarrow \text{StandardAttention}(\mathcal{W}^{(i)}, H)$
- 7: Reassemble $\{\mathbf{A}^{(i)}\}$ into $\mathbf{A} \in \mathbb{R}^{T \times c}$
- 8: **return** $\mathbf{Y} = \mathbf{A}$

LogSparse Attention [134]. This method introduces a combination of local and shifted attention, specifically tailored for time series forecasting. Unlike the previously mentioned algorithms, the representations $\mathbf{Q}$ and $\mathbf{K}$ are generated using *causal* convolutional layers, which embed local context into the attention scores. The causality is implemented by a convolution layer with a fixed kernel size and a stride of 1. The computation of $\mathbf{V}$ is performed as a point-wise convolution. After the extraction, the method computes local

¹In most cases, a relative position bias is added when calculating the attention to encode the geometric relations in the calculation.

attentions $\mathbf{A}_L$ over sliding windows of $\mathbf{K}$ and $\mathbf{V}$ with $\mathbf{Q}_t$ containing all head-specific query vectors for the input at time step t . Additionally, attention matrices $\mathbf{A}_E$ are calculated over exponential jumps, where $\mathbf{K}$ and $\mathbf{V}$ are shifted by powers of two, to cover long-range dependencies.

Algorithm 4.4 LogSparse Attention

Require: $\mathbf{X} \in \mathbb{R}^{T \times c}$, window size w , exponential jumps E , heads H

Ensure: $\mathbf{Y} \in \mathbb{R}^{T \times c}$

```

1:  $\mathbf{Q} \leftarrow \text{CausalConv1D}(\mathbf{X}, \text{kernelsize} = w)$ 
2:  $\mathbf{K} \leftarrow \text{CausalConv1D}(\mathbf{X}, \text{kernelsize} = w)$ 
3:  $\mathbf{V} \leftarrow \mathbf{X} \mathbf{W}^v$  ▷ point-wise projection
4: reshape  $\mathbf{Q}, \mathbf{K}, \mathbf{V} \rightarrow [H \times T \times d]$  ▷  $d = c/H$ 
5: Pad  $\mathbf{K}, \mathbf{V}$  on the left by  $w - 1$ 
6:  $\{\mathbf{K}^{(t)}, \mathbf{V}^{(t)}\}_{t=1}^T$  ▷ split into  $T$  overlapping windows,  $\mathbf{K}^{(t)}, \mathbf{V}^{(t)} \in \mathbb{R}^{H \times w \times d}$ 
7: for  $t = 1, \dots, T$  do
8:    $\mathbf{A}_L^{(t)} = \text{SoftMax}(\mathbf{Q}_t (\mathbf{K}^{(t)})^\top / \sqrt{d}) \mathbf{V}^{(t)}$  ▷ local attention
9:  $\mathbf{A}_L \leftarrow \text{reshape}(\text{concat}(\mathbf{A}_L^1, \dots, \mathbf{A}_L^T))$ 
10:  $\mathbf{A}_E \leftarrow 0$ 
11: for  $e = 0, \dots, E - 1$  do
12:    $\mathbf{K}_{\text{shift}}, \mathbf{V}_{\text{shift}} \leftarrow \text{Shift}(\mathbf{K}, \mathbf{V}, 2^e)$  ▷ shift by  $2^e$  along time
13:    $\mathbf{A}_E += \text{SoftMax}(\mathbf{Q} \mathbf{K}_{\text{shift}}^\top / \sqrt{d}) \mathbf{V}_{\text{shift}}$ 
14: return  $\mathbf{Y} = (\mathbf{A}_L + \mathbf{A}_E) \mathbf{W}^o$ 
    
```

Linear Attention [135]. The central idea of linear attentions is to replace the softmax kernel of the standard attention with a feature map ϕ . This adjustment enables the reordering of computations, thus eliminating the quadratic scaling associated with input length T . The feature map is typically computed using the Exponential Linear Unit (ELU) activation function [136] as

$$\phi(x) = \text{ELU}(x) + 1. \quad (4.18)$$

The algorithm of the linear attention mechanism is outlined in Algorithm 4.5. While this technique scales linearly with respect to T , it scales quadratically as the feature dimension c increases.

Algorithm 4.5 Linear Attention**Require:** $X \in \mathbb{R}^{T \times c}$, heads H , $\phi(u) = \text{ELU}(u) + 1$ **Ensure:** $Y \in \mathbb{R}^{T \times c}$

```

1:  $[Q, K, V] \leftarrow XW^{qkv}$ 
2:  $Q \leftarrow \phi(Q)$ ,  $K \leftarrow \phi(K)$  ▷ apply feature map
3: reshape  $Q, K, V \rightarrow [H \times T \times d]$  ▷  $d = c/H$ 
4: for  $t = 1, \dots, T$  do
5:    $s_t \leftarrow \sum_{i=1}^i K_i V_i^\top$  ▷  $\mathcal{O}(d^2)$ 
6:    $z_t \leftarrow 1/(Q_t^\top \sum_{i=1}^t K_i)$ 
7:    $A_t \leftarrow Q_t^\top s_t z_t$ 
8:  $A \leftarrow \text{reshape}(\text{concat}(A_1, \dots, A_T))$ 
9: return  $Y = A W^o$ 

```

Others. Various other domain-specific methods have been developed to enhance the standard attention mechanism by using different windowed calculations, similar to [131, 132], which prove advantageous for image processing tasks [137, 138]. With the growing popularity of Large Language Models (LLMs), the efficient mapping of large text sequences has gained attention. In [139], the authors introduced a dilated sliding window technique, where only a fixed number of diagonal elements of $QK^\top$ are computed. Other methods dynamically construct K and V based on the current Q to map large contexts [140]. To further leverage the computational capabilities of GPUs, certain approaches aim to fuse operations to get GPU-optimized calculations [140, 141, 142], thereby decreasing training and inference times.

All the presented techniques optimize the ordinary attention mechanism, yet they also have drawbacks. The monotonic chunked attention computes attention locally within a specified window size. However, its vanilla version does not allow cross-window connections, resulting in neighboring time steps potentially residing in different windows, and thus, relationships between them are not captured. The Swin approach addresses this limitation by enabling cross-window connections through a second attention calculation embedded in an additional transformer encoder, which shifts the local windows. As a result, a Swin Transformer must include two successive encoders. The LogSparse mechanism computes attention in local overlapping windows as well as in exponentially spaced areas around the input, leading to a more global representation. However, this representation may serve as an approximation, potentially overlooking crucial information. The authors of the linear attention approach eliminated the quadratic complexity associated with T by replacing the softmax function with a feature map. The reordering of computations allows true linear complexity but involves quadratic scaling in the feature dimension. This approach can be beneficial in domains such as text, video, or image generation, where the

number of input channels is typically fixed and limited. In contrast, time series applications can easily have a channel count in the triple digits depending on the sensors utilized. Table 4.3 presents a side-by-side comparison of the discussed mechanisms in terms of their benefits, drawbacks, and complexity regarding time and memory.

Table 4.3: Comparison of discussed attention mechanisms.

Mechanism	Advantage	Disadvantage	Complexity
Standard [53]	Full attention matrix	Quadratic in T	$\mathcal{O}(T^2 c)$
Chunked [131]	Windowed attention reduces complexity	No cross-chunk connections	$\mathcal{O}(T w c)$
Swin [132]	Cross-windows connections	Requires two successive encoders	$\mathcal{O}(T w c)$
LogSparse [134]	Local and multi-scale attention	Global context just approximated	$\mathcal{O}(T (\log T)^2 c)$
Linear [135]	True linear scaling in T	Quadratic scaling in c , approximation of the standard attention	$\mathcal{O}(T c^2)$

We end this section with an experimental setup that implements the strategies described above to compute MHA with $h = 8$ attention heads. The focus is on comparing their execution speed and memory usage for batched input time series, where each input sample $\mathbf{X} \in \mathbb{R}^{T \times c}$ has $T \in \{2^5, 2^6, \dots, 2^{11}\}$ time points and $c = 128$ channels. The chunked, Swin, and LogSparse attention mechanisms adopt a local window size of $w = 8$, with LogSparse attention employing $E = 4$ exponential jumps. The measurements, illustrated in Figure 4.7, represent the mean results across 500 iterations after ten warm-up passes. The exponential, near-linear, and linear characteristics of the different approaches are clearly visible. In terms of computational efficiency, LogSparse attention outperforms the standard attention algorithm when $t > 1,000$. However, it is still significantly slower and computationally more expensive than chunked, Swin, and Linear attention. The displayed measurements relate solely to the standalone attention calculations as outlined above. The Swin attention is typically used in a two-encoder configuration, allowing attention calculations on shifted local windows. In the sequel, we opt for the Swin attention due to its linear scalability in window T and c and due to its ability to capture wider ranges through shifted windows than chunked attention.

In addition to strategies aimed at optimizing the attention mechanism, various methods have emerged to enhance the outputs of transformer models, including simplifying the transformer encoder [143]. An analysis of different types of normalization layers is commonly conducted [144], alongside examinations of pre- and post-normalization techniques within a transformer framework [133].

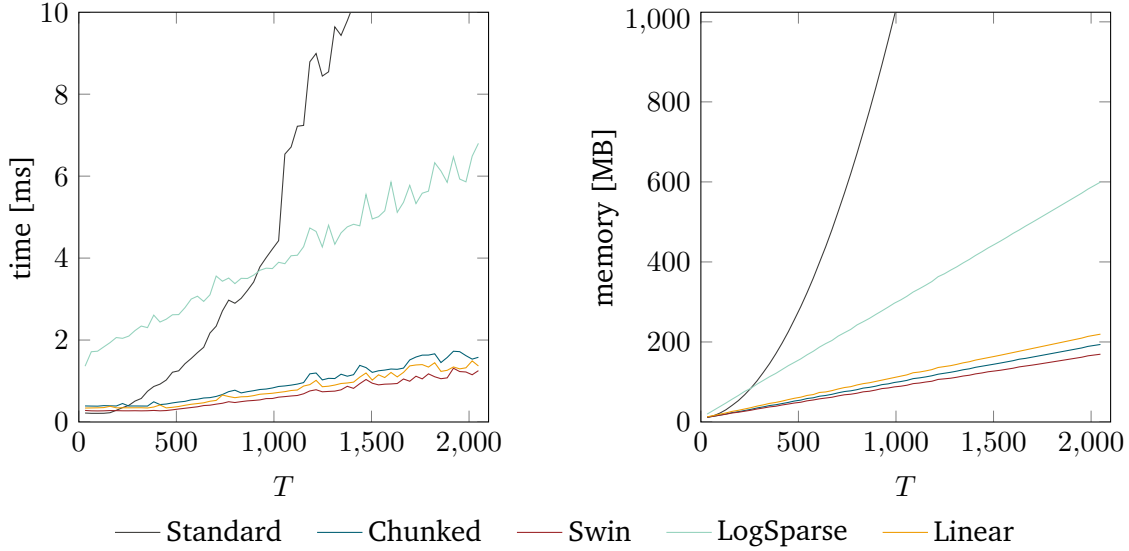


Figure 4.7: Comparison of attention algorithms regarding time (left) and memory (right) consumption.

4.3.2 Transformer U-Net for Anomaly Detection

Disclaimer: This section is widely based on the algorithm and the findings already published under the title: *Rethinking Time Series Anomaly Detection: A Scalable Transformer-Based Framework for Large Contexts* (2025) [145]. However, hyperparameters or model-specific layers may have changed.

We propose a novel transformer-based model designed to reconstruct the multivariate input time series accurately. Fundamental deep learning architectures include various variations of autoencoders that serve this purpose, as discussed in Section 2.3.2. Especially the symmetrical U-Net architecture [46], which uses additional skip connections between the encoder and decoder to merge low-resolution features from the encoder with upsampled ones, is well-suited for an accurate reconstruction. It has also been utilized in attention-based segmentation tasks before [146].

As pointed out in the previous Section 4.3.1, traditional attention mechanisms are not suitable for processing large input sequences. Instead, Swin Transformers are employed to compute attentions in smaller windows. Figure 4.8a illustrates the novel model structure, which incorporates Swin Transformer Blocks (STBs) as layers within the U-Net in the context of anomaly detection. The model, *Swin1D-AD*, employs a Swin Transformer-based architecture for time-series anomaly detection. The model’s structure is composed of several building blocks, which are discussed in detail below.

Embedding and Projection layer: The first step in our approach involves embedding the input time series $\mathbf{X}$ using a single trainable convolution layer, followed by a leaky

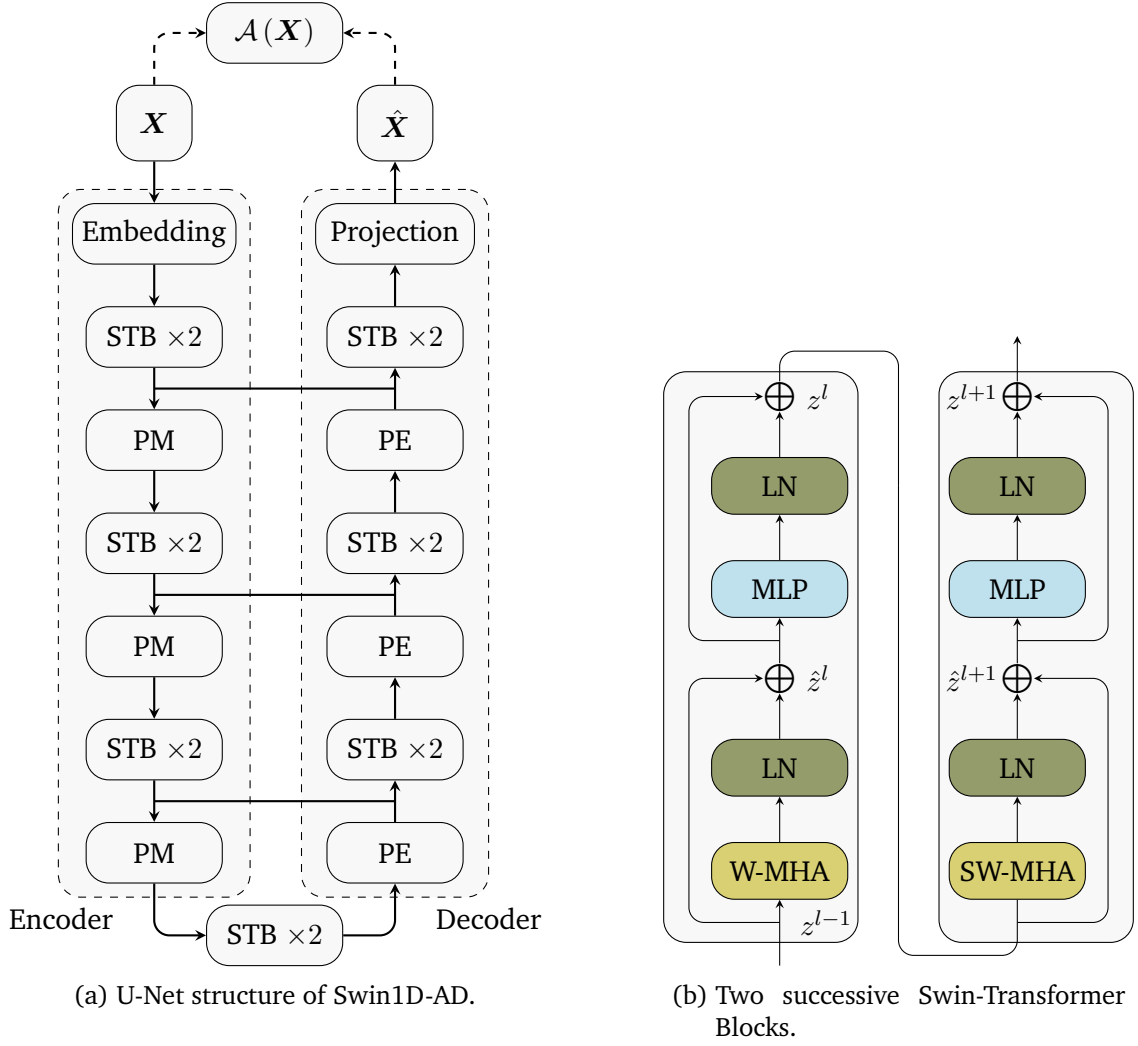


Figure 4.8: Overview of the Swin1D-AD architecture.

Rectified Linear Unit (ReLU) nonlinearity to project the features into an arbitrary dimension C . A fixed positional embedding supplements the token embedding to maintain context, as described in [53]. The projection layer serves as the counterpart and is used as the final step of the network. It includes another convolutional layer with hyperparameters appropriately configured to ensure alignment between the model's input and output shapes.

Swin Transformer Block (STB): The STBs are utilized to compute attention through the Swin mechanism. As discussed in Section 4.3.1, this attention algorithm has to be incorporated into two consecutive transformer encoders to enable windowed attention across shifted windows. These blocks are depicted in Figure 4.8b. Each block comprises a windowed attention layer, Layer Normalizations (LNs), residual connections, and a two-layer MLP with non-linearity. The blocks are designed for a time series setup in accordance

with [132], employing the *residual post normalization* technique of [133], which offers benefits when scaling up the model capacity. Based on the (shifted) window multi-head self-attention ((S)W-MHA), the successive STBs can be written as

$$\hat{z}^l = \text{LN} \left(\text{W-MHA} \left(z^{l-1} \right) \right) + z^{l-1}, \quad (4.19)$$

$$z^l = \text{LN} \left(\text{MLP} \left(\hat{z}^l \right) \right) + \hat{z}^l, \quad (4.20)$$

$$\hat{z}^{l+1} = \text{LN} \left(\text{SW-MHA} \left(z^l \right) \right) + z^l, \quad (4.21)$$

$$z^{l+1} = \text{LN} \left(\text{MLP} \left(\hat{z}^{l+1} \right) \right) + \hat{z}^{l+1}, \quad (4.22)$$

where $\hat{z}^l$ and z^l represent the normalized features from the (S)W-MHA and MLP modules of the l^{th} block, respectively.

Patch Merging (PM) and **Patch Expansion (PE)** layers: In each patch merging block, the input size is reduced by half, while the feature dimension is simultaneously increased through a linear layer. Patch expansion acts as a counterpart to patch merging, as it enlarges the input size using a single linear layer. The features extracted from the U-Net encoder are concatenated with the upsampled features generated during the patch expansion process. These fused features are processed through a linear layer, integrating the concatenated elements into a unified representation. Furthermore, LN layers are applied to each block to ensure stability and enhance convergence during training.

Swin1D-AD follows a reconstruction-based approach. Therefore, the objective in our model is expressed by minimizing the reconstruction loss as the MSE

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^N \left(\mathbf{X}_n - \hat{\mathbf{X}}_n \right)^2, \quad (4.23)$$

where N is the number of samples in the training batch. The reconstruction error represents the difference between the original time series data and the data reconstructed by the model. This error is a critical indicator of the model's proficiency in learning the normal patterns within the data.

The reconstruction error is highly effective in highlighting anomalies. When the model encounters an anomaly during inference, the reconstruction error typically increases significantly, thereby facilitating the identification of unusual patterns. While MSE is sensitive to large errors, it is also robust to minor fluctuations or noise in the data. This balance ensures that the model does not overreact to insignificant variations, focusing instead on

substantial anomalies. Our model uses the MSE also at a point level during inference to get the anomaly scores $\mathbf{a} \in \mathbb{R}^{T \times 1}$ for every time step as

$$\mathbf{a} = \mathcal{A}(\mathbf{X}) = \frac{1}{c} \sum_{i=1}^c \left(\mathbf{X}_i - \hat{\mathbf{X}}_i \right)^2, \quad (4.24)$$

where the reconstruction error is averaged across the channels.

4.3.3 Hyperparameter Settings

The model was trained with various input window sizes T ranging from 64 to 1,024 with a stride of 64, thereby showing its reconstruction capabilities. To perform a systematic grid-search over the C hyperparameter, 17 values equally distributed between 8 and 256 were evaluated. For each C and T value, the model was trained and tested under the same conditions and remaining hyperparameter configurations, enabling controlled assessment of performance sensitivity to the penalty strength. In the STBs, $h = [4, 8, 8, 16]$ heads were used for the MHA modules for down- and upstream, respectively. The training was performed utilizing the AdamW optimizer [130] for 10 epochs using the same parameter setting as before.

To evaluate the different combinations of T and C , a sample test time series from a publicly available anomaly detection dataset is used, namely SMD machine-1-6². Figure 4.9 shows the MSE between the original and reconstruction across all channels for the different settings under consideration of the corresponding label: Figure 4.9a shows the errors of normal data points, whereas Figure 4.9b shows the mean error at time steps when an anomaly is present. The discrepancy in the reconstruction error of normal and anomalous time steps should be significant, while the reconstruction score of normal data should be low. For this example, the Swin1D-AD model was used with a local window size for the attention mechanism of $w = 8$.

A small value for C leads to a more compact model, quicker training, and lower inference times. It also forces the model to extract a significant amount of information during the first embedding step and then narrow the information further down to the lower layer of the U-Net, resulting in high reconstruction errors. A large value for C , on the other hand, allows the model to retain all the essential information from the input time series, enabling accurate reconstruction even if an anomaly is present. According to the figures, the window size T has no impact on the models reconstruction capabilities in this example. The reconstruction errors remain relatively constant when $C > 128$, suggesting that setting $C = 128$ appears to result in the optimal balance between reconstruction accuracy and model size. This value will be adopted in subsequent experiments.

²The dataset will be introduced in Chapter 6

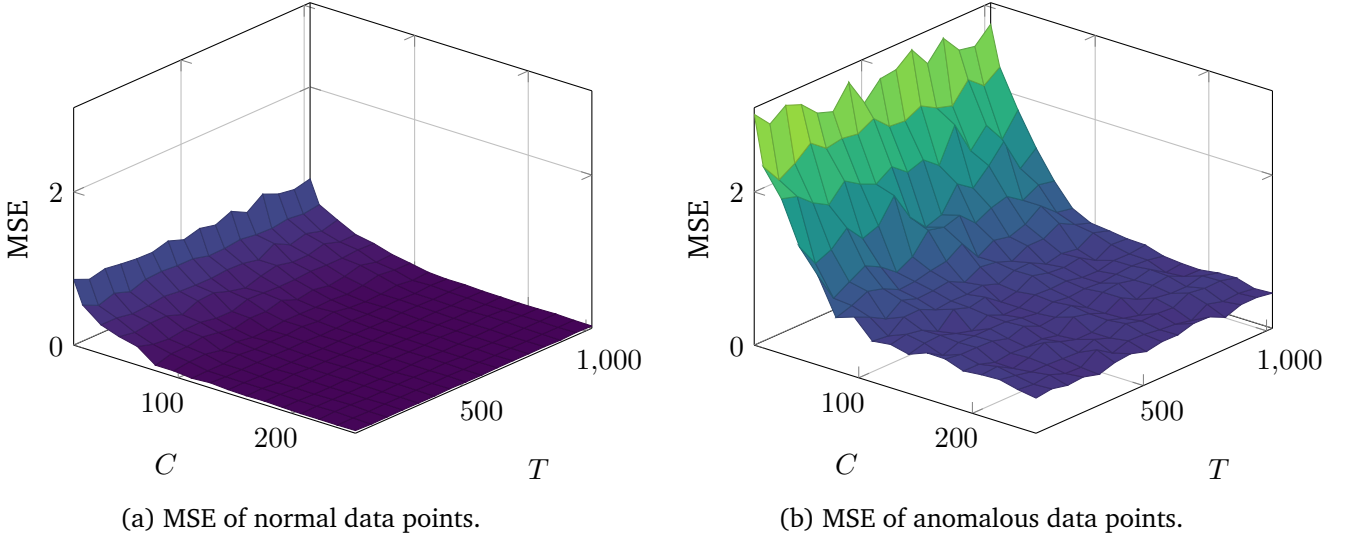


Figure 4.9: Impact of the model dimension and input window size on its reconstruction capabilities. The color code indicates the value of the MSE and is shared between the two sub-figures.

A critical parameter in the Swin1D-AD model for finding temporal correlations is the local window size w . Experiments with varying T and w values have not shown either positive nor negative effects for this time series, as the reconstruction accuracy stays at the same level.

The results of the demonstrated experiments rely heavily on the time series used, its temporal correlations, and the anomalies present. Sections 6.1 and 7.1 will provide a deeper understanding of the used data by analyzing patterns and statistics about publicly available and real-world datasets.

4.4 Diffusion Model

A traditional diffusion model is designed to remove the noise component from an input sample during training. During evaluation and inference, a new sample from within the data distribution is generated from normally distributed noise [48]. By conditioning the model, it can be guided to the desired result, such as a specific sample class that should be displayed.

The functionality of the ordinary diffusion model was introduced in Section 2.3.3, while different conditioning techniques tailored to time series anomaly detection were presented in Section 4.4. This chapter examines the impacts of various techniques within the forward and backward processes on the reconstruction result and discusses possible improvements with respect to the ordinary DDPM.

To find the most suitable configuration of the diffusion model, the forward process and the backward process are discussed separately in Sections 4.4.1 and 4.4.2. Section 4.4.3 presents experiments with various configurations, identifying the one used in the following chapter. Section 4.4.4 summarizes the findings of our implemented diffusion model.

4.4.1 Forward Process

In the forward process, an input sample is gradually perturbed by adding Gaussian noise incrementally. With an increasing diffusion step k , the resulting sample becomes a noisier version of the original, thereby following a controlled schedule. This process can be written as

$$\mathbf{X}^k = \sqrt{\bar{\alpha}_k} \mathbf{X}^0 + \sqrt{1 - \bar{\alpha}_k} \epsilon, \quad (4.25)$$

as already detailed earlier in Equation (2.34). As k increases, the signal-to-noise ratio (SNR) decreases, and the data gradually becomes disrupted until it resembles pure noise. This process forms the foundation of diffusion models and serves as the basis for subsequent modifications to the forward dynamics.

Beta Scheduler

The type of scheduler and its parameters play a crucial role in determining the noise level of $\mathbf{X}^k$ at each stage. The following commonly used algorithms are presented to compute $\{\beta_k\}$. Figure 4.10 visualizes the corresponding $\sqrt{\bar{\alpha}_k}$ values across $K = 100$ diffusion steps when the different schedulers are applied. The different schedulers are introduced below.

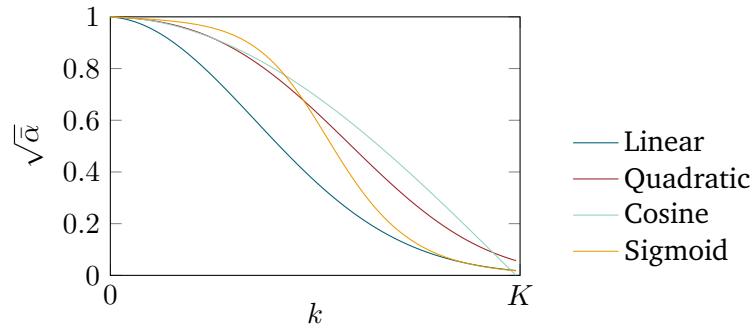


Figure 4.10: The values of $\sqrt{\bar{\alpha}}$ with different schedulers.

A straightforward approach is to use a linear scheduler, as employed in [48], where the value of β increases linearly with k . This scheduler provides a uniform noise accumulation from a given start value to an end value (typically $\beta_{\text{start}} = 0.0001$ and $\beta_{\text{end}} = 0.02$). The pseudocode is provided in Algorithm 4.6.

Algorithm 4.6 Linear Beta Scheduler

```

1: procedure LINEARBETASCHEDULE( $K, \beta_{\text{start}}, \beta_{\text{end}}$ )
2:   for  $k = 1$  to  $K$  do
3:      $\beta_k \leftarrow \beta_{\text{start}} + k \frac{\beta_{\text{end}} - \beta_{\text{start}}}{K-1}$ 
4:   return  $\{\beta_1, \beta_2, \dots, \beta_K\}$ 

```

A different scheduler was introduced in [147]. Their quadratic approach ensures a slower noise growth at the early diffusion steps and a faster noise growth in later steps. Such a strategy can enhance the overall performance of diffusion models by striking a balance between retaining essential signal characteristics and the efficiency of the diffusion process. The pseudocode can be found in Algorithm 4.7.

Algorithm 4.7 Quadratic Beta Scheduler

```

1: procedure QUADRATICBETASCHEDULE( $K, \beta_{\text{start}}, \beta_{\text{end}}$ )
2:   for  $k = 1$  to  $K$  do
3:      $\tau \leftarrow k/(K-1)$ 
4:      $\beta_k \leftarrow \beta_{\text{start}} + (\beta_{\text{end}} - \beta_{\text{start}})\tau^2$ 
5:   return  $\{\beta_1, \beta_2, \dots, \beta_K\}$ 

```

With the cosine scheduler [108], as displayed in Algorithm 4.8, the signal is also preserved longer in early timesteps, which often leads to better sampling quality and faster convergence. While this approach does not need start and end values, an additional parameter s is used for numeric stability. The authors of [108] choose to set $s = 0.008$ after the pixel bin size. In a time series context, s can be viewed as a parameter that controls the amount of residual variance remaining at the end of the diffusion process.

Algorithm 4.8 Cosine Beta Scheduler

```

1: procedure COSINEBETASCHEDULE( $K, s$ )
2:   for  $k = 1$  to  $K$  do
3:      $f_k \leftarrow \cos^2 \left( \frac{(k/K+s)}{1+s} \frac{\pi}{2} \right)$ 
4:   for  $k = 1$  to  $K$  do
5:      $\bar{\alpha}_k \leftarrow \frac{f_k}{f_0}$ 
6:      $\beta_k \leftarrow 1 - \frac{\bar{\alpha}_k}{\bar{\alpha}_{k-1}}$ 
7:   return  $\{\beta_1, \beta_2, \dots, \beta_K\}$ 

```

The sigmoid-based scheduler, as displayed in Algorithm 4.9, is a more stable alternative during training [148] and can be found in various implementations of diffusion models³. Especially the smooth sigmoid start ensures that early diffusion steps slightly transforms the data, which enables that important structures in the data that are relevant for downstream tasks are retained. The additional hyperparameter s represents the steepness of the sigmoid curve, making the algorithm more controllable.

Algorithm 4.9 Sigmoid Beta Scheduler

```

1: procedure SIGMOIDBETASCHEDULE( $K, \beta_{\text{start}}, \beta_{\text{end}}, s$ )
2:   for  $k = 1$  to  $K$  do
3:      $\tau \leftarrow 2 \frac{k}{K-1} - 1$   $\triangleright$  Normalized timestep in  $[-1, 1]$ 
4:      $z_k \leftarrow \frac{1}{1+e^{-s\tau}}$ 
5:      $\beta_k \leftarrow z_k(\beta_{\text{end}} - \beta_{\text{start}}) + \beta_{\text{start}}$ 
6:   return  $\{\beta_1, \beta_2, \dots, \beta_K\}$ 

```

None of the mentioned algorithms is exclusively tailored to data structures, but is commonly used across domains and tasks. Lee et al. [149] designed an adaptive noise scheduler, thereby focusing on time series generation and forecasting. With their approach, the most suitable noise scheduler for the specific dataset can be determined beforehand by measuring the non-stationary behavior of the samples over all diffusion steps. In their approach, they utilized the autocorrelation function to calculate the non-stationary statistics and rank the schedulers accordingly. This technique requires analyzing the dataset before training, but it makes it easier to find appropriate hyperparameters. In most tests on the datasets discussed later, the sigmoid and the quadratic beta scheduler perform slightly better than the other mentioned techniques. The values for $\sqrt{\alpha}$ over K diffusion steps with different beta schedulers are shown in Figure 4.10.

In the context of image-based diffusion models, the input resolution of $\mathbf{X}^0$ is another critical parameter [150, 151] when setting up the scheduler. The studies focus on different scaling factors for the image. When images are upscaled, the influence of the scheduler on the noise level is reduced. With time series data, the window size is not important for the scheduler, as a larger window size does not result in interpolated or replicated data points, but rather successive time steps added.

Impact of the Diffusion Steps K and the Sampling of k

For traditional DDPMs, it was shown that generated images are better when having a large number of diffusion steps ($K = 1,000$ in [48, 107]). For time series data, it has been

³<https://github.com/rosikand/ddpm-pytorch>,
<https://github.com/ron-taieb/SchedNoise-Diffusion>,
<https://github.com/lucidrains/denoising-diffusion-pytorch>

shown, e.g., in [114, 115, 149], that this is not necessarily the case, and suitable results can be generated with $K \leq 200$ diffusion steps. The reduced number of steps during generation has the advantageous side effect of reduced sampling time and, therefore, faster inference.

In typical training procedures, the variable diffusion step k is sampled uniformly from the range $[1, K]$ [48]. This approach ensures that the model is exposed to all noise levels equally, thereby effectively mitigating the risk of overfitting to specific values of k . However, a uniform sampling strategy might not be the optimal approach for achieving fast convergence, as different noise levels have varying variances in stochastic gradients.

Kim et al. [152] introduced an adaptive non-uniform sampling strategy that focuses on diffusion steps with a high variance and prioritizes these during the training process. This approach trains a policy using a REINFORCE-like gradient, with the reward being the difference in loss before and after a diffusion model update at the sampled timestep k . While these additional steps are time-consuming during training, the approach does not affect inference times. Initial experiments on a time series dataset reveal that the highest variances are observed in samples with both low and high noise levels. Figure 4.11 shows the distribution of k in epoch 50 when the adaptive sampling is applied on an ordinary DDPM for time series reconstruction. For $K = 100$, a uniform sampling approach results in each diffusion step occurring 1% (indicated by the red line) of the time. The shown policy sampling predominantly selects k values close to 0 and near K , suggesting these are the regions where the model can gain the most learning.

Contrary to this adaptive sampling technique, Jiang et al. [153] introduced a more straightforward approach of non-uniform sampling, where k is sampled more often at higher noise levels, as it is assumed that the model has more difficulty predicting the added noise there.

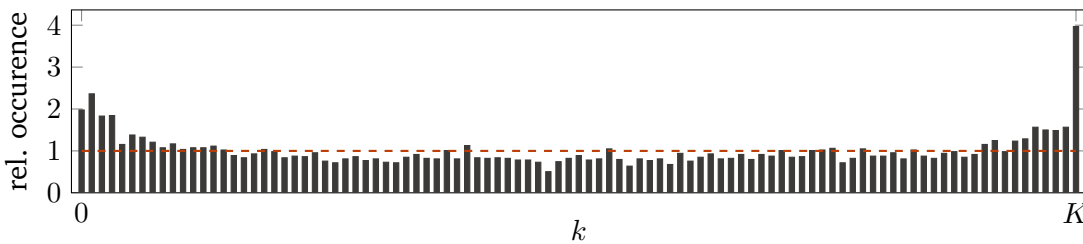


Figure 4.11: Distribution of k in epoch 50 during training when applying the adaptive non-uniform sampling strategy and $K = 100$ in contrast to uniform sampling, indicated in red.

Yet another technique for an improved training based on the diffusion steps was proposed by Hang et al. [154]. Instead of changing the sampling algorithm of k , they weight

the loss according to the SNR in the diffusion step. The loss of a larger k value is thereby weighted higher, due to the increased difficulty level for correct predictions.

Experiments have shown that a different weighting is the most powerful technique in this work. While learning an additional policy is more accurate, it comes with additional overhead and does not dramatically improve the reconstruction result.

Impact of the Noise Type ϵ

During the forward process, the input is corrupted by ϵ . Typically, ϵ is drawn from the Gaussian distribution $\mathcal{N}(0, \mathbf{I})$. More generally, a corrupted sample can be created by iteratively degrading the original. The corruption need not be Gaussian but can be represented by various functions, such as blurring, masking, or downsampling when the input is an image, as shown in [155].

While masking and downsampling can be directly transferred to the time series domain, blurring corresponds to a smoothing operator, such as a convolutional operator with a fixed kernel. Those techniques have the effect that the sample at the maximum corruption state $\mathbf{X}^K$ is not an approximation of normally distributed noise, so the generation process of a sample cannot start at $\mathcal{N}(0, \mathbf{I})$ but with the sample corrupted by the forward process.

Summarizing the Forward Process

This subsection highlighted the complexity of the forward process of diffusion models, particularly in a time series context, where techniques developed for image data are not suitable and have not been adequately investigated.

Each part can be modified, and different configurations were discussed. Various schedulers to obtain β were presented that determine the SNR at different forward steps. The total number of forward steps K is held lower when working with time series data, in contrast to image data. During training, the diffusion step k is typically sampled uniformly to cover the complete range of sample degradations. However, other methods exist that try to sample k based on the difficulty for the model. In most cases, diffusion models disrupt the data by injecting Gaussian noise; however, other techniques have also proven suitable, at least in the image domain.

Despite the different configurations and selections of techniques, the forward process always aims to degrade an input sample incrementally until little or no information remains. Mostly $\mathbf{X}^K$ approximates a Gaussian distribution. Figure 4.12 shows the mean and standard deviation of a batch of randomly chosen training samples across the K diffusion steps. Here, the ordinary DDPM was used to depict each $\mathbf{X}^k$ with a quadratic beta scheduler, as described in Algorithm 4.7, and $K = 100$ diffusion steps were employed. While this figure does not reveal specific values for particular samples, it shows the gradual degradation of the samples, as their mean values and standard deviations converge

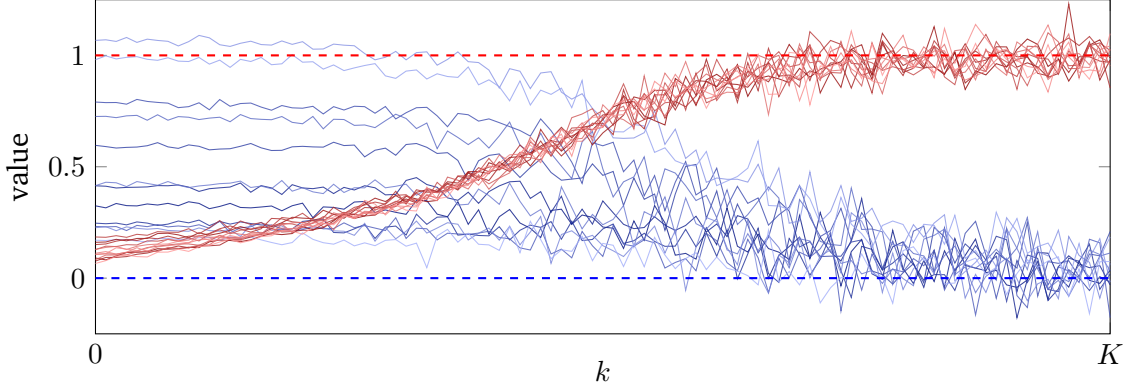


Figure 4.12: Visualization of the **mean values** and **standard deviations** of different training samples when applying the DDPM forward step at different diffusion steps. The dashed lines represent the target distribution at $k = K$.

towards 0 and 1 respectively, indicated by the dashed lines. The diffusion model used in the following also utilizes this configuration. All design choices for performing anomaly detection are summarized in Section ??.

4.4.2 Conditioned Backward Process

The backward process has to be designed in line with the forward process, utilizing the same schedulers, techniques, and approaches. For traditional DDPMs, the sampling of $\hat{\mathbf{X}}^0$ can be defined as a learned Gaussian transition in Equation (2.35):

$$p_{\theta}(\mathbf{X}^{k-1} | \mathbf{X}^k) = \mathcal{N}(\mathbf{X}^{k-1}; \mu_{\theta}(\mathbf{X}^k, k, \mathbf{c}), \beta_k \mathbf{I}), \quad (4.26)$$

Sampling scheme

Besides the stochastic, Markovian sampling of DDPMs, other strategies exist. A popular method is Denoising Diffusion Implicit Model (DDIM) [156], which keeps the same training objective and marginals $q(\mathbf{X}^k | \mathbf{X}^0)$ as DDPM but defines a non-Markovian family of reverse processes whose updates can be made deterministic. In practice, DDIM uses

$$\hat{\mathbf{X}}^0 = \frac{1}{\sqrt{\bar{\alpha}_k}} \left(\mathbf{X}^k - \sqrt{1 - \bar{\alpha}_k} \epsilon_{\theta}(\mathbf{X}^k, k, \mathbf{c}) \right) \quad (4.27)$$

to predict $\hat{\mathbf{X}}^0$ from $\mathbf{X}^k$ and the update to a previous $k' < k$ can generally be written as

$$\hat{\mathbf{X}}^{k'} = \sqrt{\bar{\alpha}_{k'}} \hat{\mathbf{X}}^0 + c_{k'} \epsilon_{\theta}(\mathbf{X}^k, k, \mathbf{c}) + \sigma_{k'} \epsilon, \quad (4.28)$$

where $c_k = \sqrt{1 - \bar{\alpha}_{k'} - \sigma_k^2}$ and $\sigma = \eta \sqrt{\frac{1 - \bar{\alpha}_{k'}}{1 - \bar{\alpha}_k}} \sqrt{1 - \frac{\bar{\alpha}_k}{\bar{\alpha}_{k'}}}$ with a controllable hyperparameter η that can remove the noise term (with $\eta = 0$) and yield a deterministic sampler. Setting $\eta = 1$ results in the DDPM sampling process. Due to its faster sampling and the ease of replacement with the DDPMs module, DDIM is a popular sampling alternative. However, it has the drawback of underexplored stochasticity.

Conditions

Several possibilities for conditioning a model towards a desired output, specifically an anomaly-free reconstruction of the input, have been discussed in Section 3.2.3. The presence of a conditioning variable, denoted as c , is necessary for the sampling process since the model typically starts the generation from pure noise [48], which necessitates some form of learned knowledge regarding the target output.

Despite the time series-specific conditions previously addressed, a popular method for anomaly detection in images is to use an encoded version of the input [29, 157] as a condition to enable reconstruction. To apply this technique to the time series domain, an encoder is required that can efficiently compress the input into a latent space. For this purpose, a straightforward convolutional autoencoder as used in the generator of the AE-GAN was utilized to derive the latent representation z of the input. Although its architecture is relatively simple and therefore not as powerful as, e.g., the Swin1D-AD, it retains the capacity to extract meaningful features that accurately reflect the input. To be able to concatenate the latent representation with the noisy input, a small MLP is utilized to adjust feature dimensions $c = \text{MLP}(z)$. The undercomplete autoencoder can be trained before or jointly with the diffusion U-Net by minimizing the reconstruction loss. During inference, the latent representation must be calculated only once for each sample and can be reused for every diffusion step. The autoencoder reconstruction is only used for training the model parameters. Due to its limited capacity, the reconstruction resembles the input signal to which a high-pass filter has been applied. An example of an input signal and the output of the convolution-based autoencoder is visualized in Figure 4.13. As the autoencoder is trained on data representing the nominal state only, anomalies cannot be effectively mapped into the latent representation.

To address this knowledge gap, Chen et al. [158] introduced a self-conditioning technique to stabilize the model predictions. The core idea of this approach is to include the $\hat{X}^0$ derived from X^{k+1} as an additional input to the model for some samples. The samples are chosen randomly with a 50% probability. For the remaining 50% of the training samples, the self-condition is set to $\mathbf{0}$. This strategy ensures the model learns to operate both with and without self-conditioning.

When employing various conditioning methods through concatenation, the total input size can increase significantly, particularly when each channel is conditioned indepen-

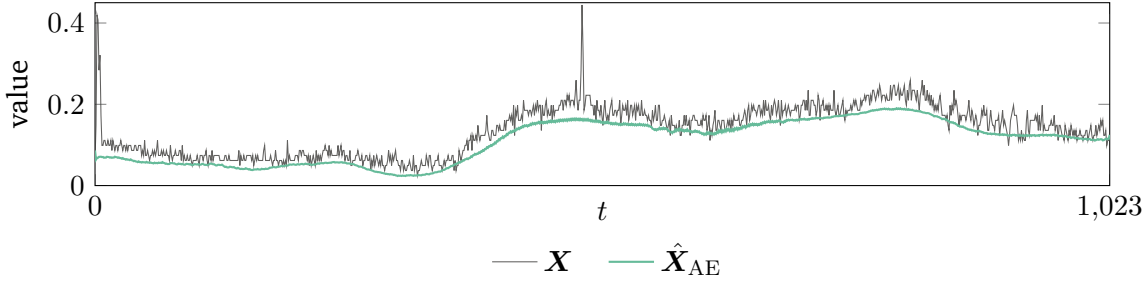


Figure 4.13: Original sample and the reconstruction of the simple autoencoder.

dently, e.g., by masking. The number of channels in a multivariate time series data alone can thereby be manifold, depending on the sensors used. In contrast, the channel count in image data is typically constrained by the color space. The potential information gain from using, e.g., the self-conditioning technique can lead to high memory and time consumption during training and inference.

Different conditioning techniques were discussed that can be integrated into standard diffusion networks. Experiments are carried out in Section ??.

Model Design

The different conditioning techniques require minimal changes to a traditional U-Net. The additional inputs are typically concatenated in the channel dimension with the noisy input, resulting in more input channels for the first layer. The conditioned U-Net utilized is visualized in Figure 4.14 along with the different building blocks. The *flow* of the input X^k is displayed by black arrows and the *flow* of the diffusion step k by orange arrows.

The noisy data is firstly concatenated with its condition $[X^k, c]$ as done in [118, 159]⁴. The embedding layer for the input consists of a single convolutional layer that transforms the input to an arbitrary dimension. Each down- and upsampling block (DownB/UpB) consists of two modified ResNet blocks [113] followed by an attention calculation. Convolutional or transposed convolutional layers are used to realize upsampling and down-sampling. Between the encoder downstream and the decoder upstream, there are skip connections that project the features directly to the decoder at different levels. Furthermore, a layer consisting of a ResNet block, attention calculation, and another ResNet block forms the bottleneck of the model, where the data is compressed the most. The MHA module displayed is used as a placeholder for various attention calculations, as introduced in Section 4.3.1. In our particular setting, we use the *linear attention* [135] in the down- and upstream, and the *standard attention* [53] in the bottleneck layer. We employ a causal convolutional layer as in [134] to compute the queries and keys for the attention mechanisms. The current diffusion time step k is transformed by a sinusoidal embedding as

⁴Self-conditioning is omitted here, which results in further concatenation.

proposed in [53], followed by an MLP. The embedded diffusion step condition is injected into each ResNet block as displayed in Figure 4.14c. A two-layered MLP is used to transform the input to the correct feature dimension in each layer to further scale and shift the input. This allows the model to adapt the amplitude and baseline of each feature-map channel conditioned on the noise level, giving the network the flexibility to adjust normalization and feature suppression/amplification depending on the input noise level [160]. The projection layer at the end of the U-Net consists of a single ResNet block and a final convolutional layer to match the dimensions of $\mathbf{X}^k$ and the predicted noise $\hat{\epsilon}$.

Incorporating c into each ResNet layer, analogous to the incorporation of k , did not result in a more accurate prediction in the experiments.

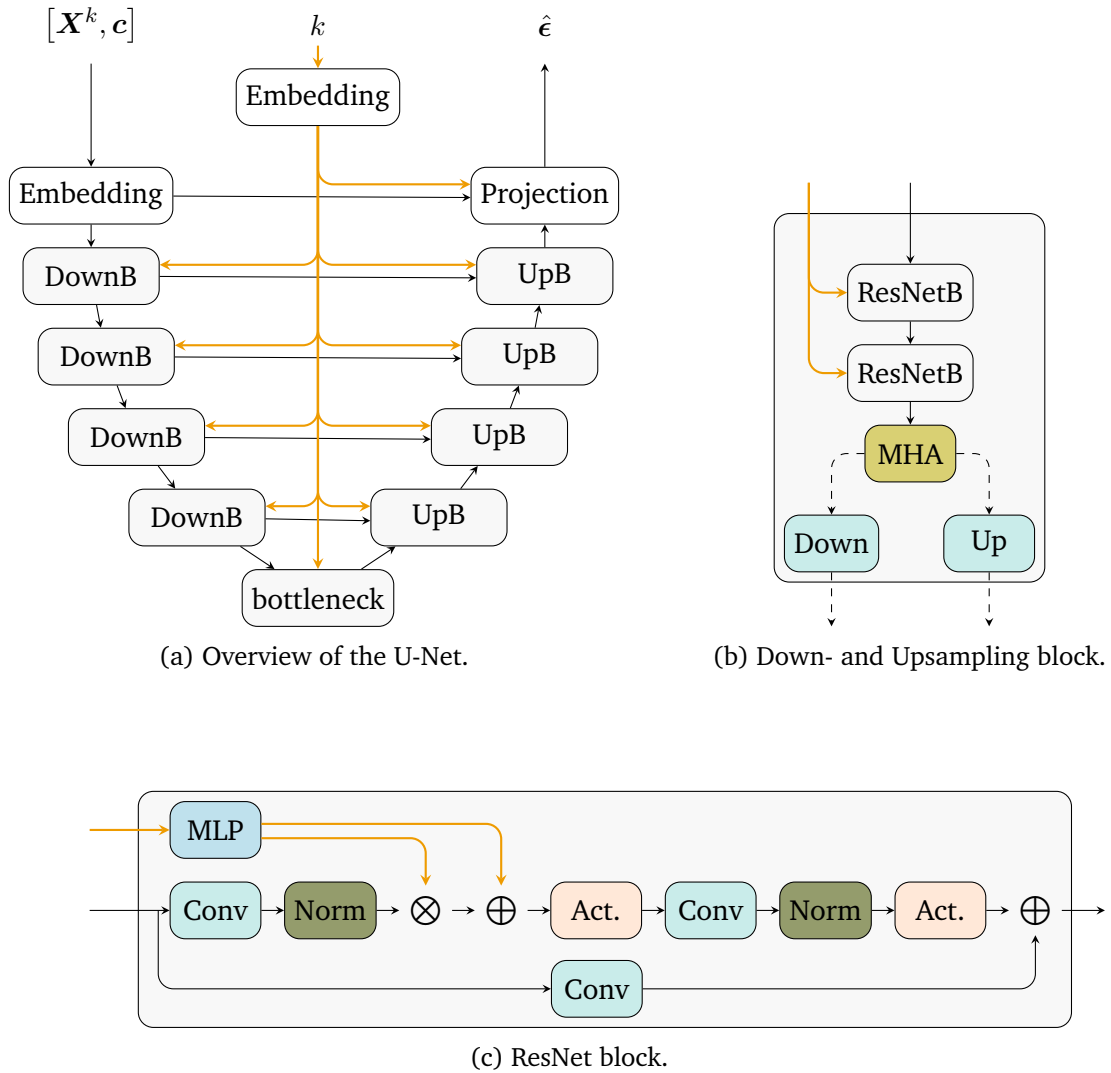


Figure 4.14: U-Net structure used in the diffusion model.

Summary and Conclusion

The components of the backward process are designed to reconstruct a normal-looking sample, enabling anomaly detection.

A crucial aspect of the model’s performance is its conditioning. While various techniques were explored earlier in Section 3.2.3, such as incorporating the masked signal as an additional input [114], we opted to utilize the latent representation of the input samples as the model’s informational foundation. The condition has to be carefully selected to be sufficiently large to capture sufficient information about the sample, while simultaneously remaining small enough to exclude any details regarding anomalies. An additional condition can be used when enabling self-conditioning, which showed improved generation results in the past.

In contrast to other approaches that utilize transformer-only architectures to predict the noise at each diffusion step [114, 116], our model employs the U-Net architecture as in traditional diffusion models. This choice allows for flexibility in handling varying input sizes and thereby leveraging extensive context. As a backbone in diffusion models, the U-Net has proven to be effective in capturing both global structures and fine details of the input [161]. The used linear attention ensures faster processing speeds compared to the standard attention module, which is essential for diffusion models, given that the model must be called multiple times to generate a sample. Additionally, this approach enables a wider context window compared to the convolutional layers typically used in U-Nets. Previous experiments in Section 4.3.1 have already illustrated the advantages of this attention module in terms of time and space complexity when testing various input sizes.

4.4.3 Experimental Studies

Different configurations were implemented and tested in an experimental environment using the publicly available data mentioned. The input window size is held constant at $T = 1,024$ throughout the experiments. This section is split into the training process and the validation process of the configurations.

Diffusion Model Training

All presented model configurations aim to minimize the reconstruction error in terms of the MSE. Figure 4.15 shows the loss curves of several trainings with varying configurations.

The base model utilizes self-conditioning with $K = 100$ DDPM steps, which are non-uniformly sampled as described in [154], and the quadratic scheduler to obtain β values. The scheduler start and end points are selected depending on K to transform the input

signal correctly. The discrete diffusion time step k is replaced with a generalization to a continuous $k \in [0, 1]$ as done in [147]. For $K = 100$, similar statistics as in Figure 4.12 can be obtained with $\beta_1 = 0.0001$ and $\beta_K = 0.2$. The size of the encoded signal, which is used as a condition, is set to 64. The base model is trained for 50 epochs with a learning rate of 0.0003 with the AdamW optimizer [130] to predict the noise $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, as with conventional diffusion models. For the other displayed loss curves, one of the mentioned values was changed, as indicated in the respective legend and caption. The subfigures are described in more detail below.

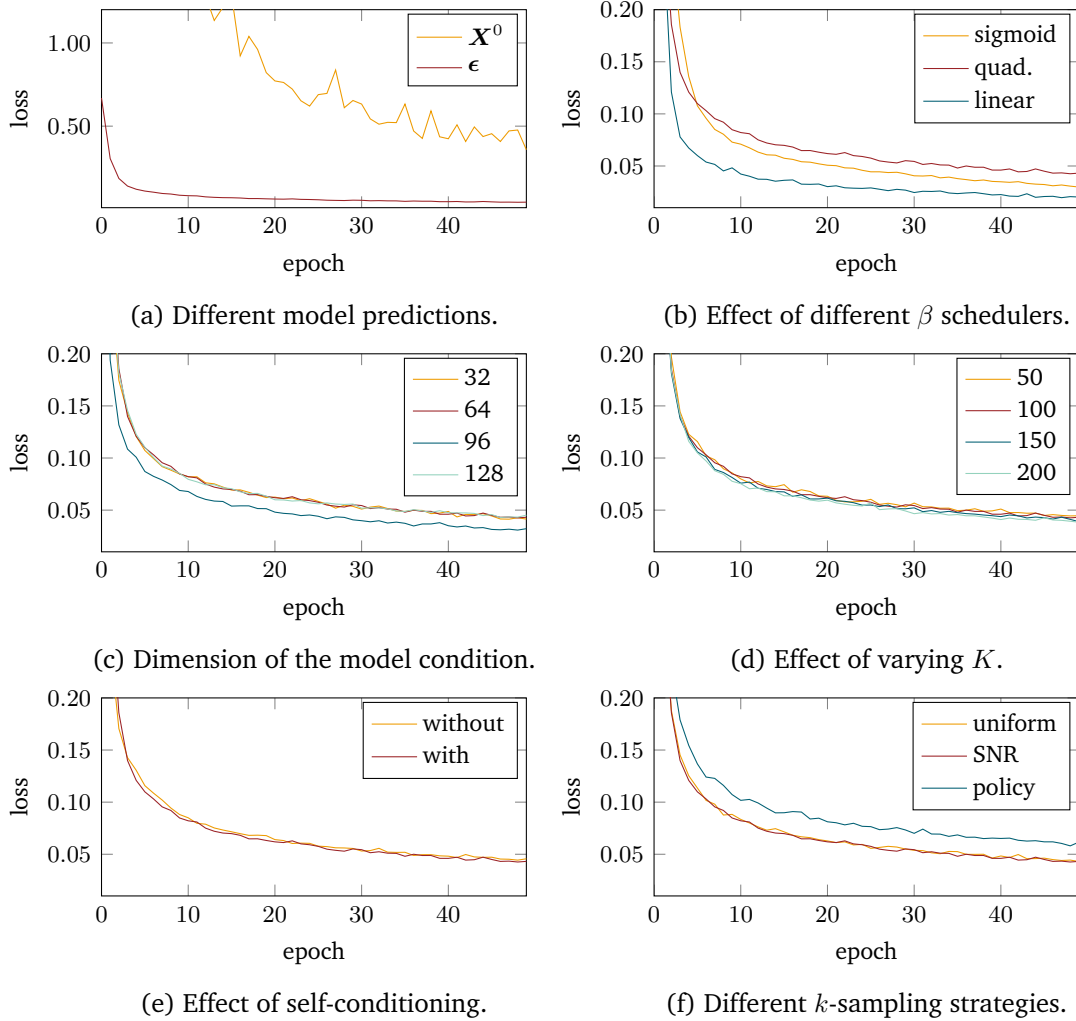


Figure 4.15: Impact of different configurations on the training process represented by training losses (as MSE) over 50 epochs of training with respect to the **base** implementation.

The loss curves of the different model configurations are mostly similar, monotonically declining, and converging. Figure 4.15a displays the loss curves when the model is trained

to predict ϵ and $\mathbf{X}^0$ directly. The model is struggling during training, indicated by the more volatile loss curve.

An important design choice is the selection of the β scheduler and its configuration. The behavior of different discussed schedulers is visualized in Figure 4.15b, thereby using the same range of values. The cosine scheduler is omitted here, as the training did not converge.

When encoding the signal into the latent representation $\mathbf{z}$, the size of the representation plays a crucial role. A smaller representation reduces the likelihood of mapping potential anomalies. However, if the representation is too small, it fails to accurately capture the signal, making it inefficient for use as a condition in the diffusion backbone model. Figure 4.15c shows the size of the encoded signal, which is used as a condition for the reverse diffusion process.

The number of diffusion steps has no particular influence on the training process (Figure 4.15d), as a continuous time step is used in the experiments. However, it is essential to note that the β schedulers had to be adjusted for these experiments, as the scheduler is dependent on K .

Despite the theoretical benefits of the self-conditioning technique, it did not show practical advantages in the loss curves, as can be seen in Figure 4.15e. A potential explanation for this is that the former prediction, which led to the self-condition, is too imprecise.

The various sampling strategies for obtaining k during training were presented earlier and tested. The corresponding loss curves are shown in Figure 4.15f. There is no visual difference in the loss curves when comparing the uniform sampling and the weighted sampling according to the SNR.

The loss curves show the mean reconstruction capabilities of the diffusion model from each diffusion step k in the respective configuration. However, the displayed loss curves do not provide any information about the model performance during sampling when performing the complete reverse process, starting with a Gaussian distribution.

We evaluated the presented configurations on the appropriately labeled testing dataset and measured the mean reconstruction errors for both normal and anomalous time steps. The results are given in Table 4.4. For a better overview, the table is structured similarly to Figure 4.15.

Diffusion Model Sampling

To validate the different configurations, the reconstruction capabilities were investigated using the corresponding test data. Starting with $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the model iteratively refines its output across the diffusion steps. For each step, we calculate a reconstruction loss that indicates the refinement of the reconstruction as the diffusion step k decreases.

Table 4.4: MSE of normal and anomalous time steps in the testing data. a) model prediction, b) β scheduler, c) dimension of the condition, d) varying K , e) self-conditioning, f) k -sampling.

		normal	anomaly			normal	anomaly
(a)	$\mathbf{X}^0$	0.018	0.062	(b)	sigmoid	0.171	0.206
	ϵ	0.015	0.052		quad.	0.015	0.052
					linear	0.093	0.160
(c)	32	0.014	0.049	(d)	50	0.030	0.058
	64	0.015	0.052		100	0.015	0.052
	96	0.013	0.045		150	0.015	0.049
	128	0.013	0.046		200	0.028	0.061
(e)	with	0.015	0.052	(f)	uniform	0.014	0.052
	without	0.015	0.048		SNR	0.015	0.052
					policy	0.015	0.047

Figure 4.16 shows the reconstruction errors for two arbitrary samples of the testing dataset, using $K = 100$ diffusion steps. The top graph shows the MSE values, whereas the bottom graphs show a modified cosine similarity between the original and predicted signal. In one of these samples, an anomaly is present, and the reconstruction curve converges on a higher level. Using a similarity measurement, such as DTW [17] or cosine similarity, between the input sample and the reconstructions instead of the MSE, results in a similar error curve across the diffusion steps but with less distance, depending on the size and shape of the anomaly, as can be seen in the lower graph. Here, the cosine similarity was used as

$$\text{cos_sim}(\mathbf{X}, \hat{\mathbf{X}}) = \frac{\langle \mathbf{X}, \hat{\mathbf{X}} \rangle}{\|\mathbf{X}\|_2 \|\hat{\mathbf{X}}\|_2}. \quad (4.29)$$

Under the formulation $1 - \text{cos_sim}(\cdot, \cdot)$, a perfect alignment between the input representations yields a distance of 0, while increasing values indicate progressively poorer alignment.

The reconstruction error at early diffusion steps is low, as the displayed values represent the mean across all channels, of which some are constants at zero. The curves representing the similarity measure are clearly separated. Focusing on a single channel eliminates the effect of channels that are constant or do not show odd behavior at the labeled anomaly. When utilizing such a channel, the error when generating from $\mathbf{X}^K$ is significantly higher, as displayed in Figure 4.17.

However, there is no improvement in the reconstruction capabilities after generating $\hat{\mathbf{X}}^0$ from $\mathbf{X}^{70}$ in all the shown curves, regardless of the observed number of channels, suggesting that the reconstruction does not get more accurate afterwards. The remaining diffusion steps can be skipped, which saves time during inference. Jiang et al. have

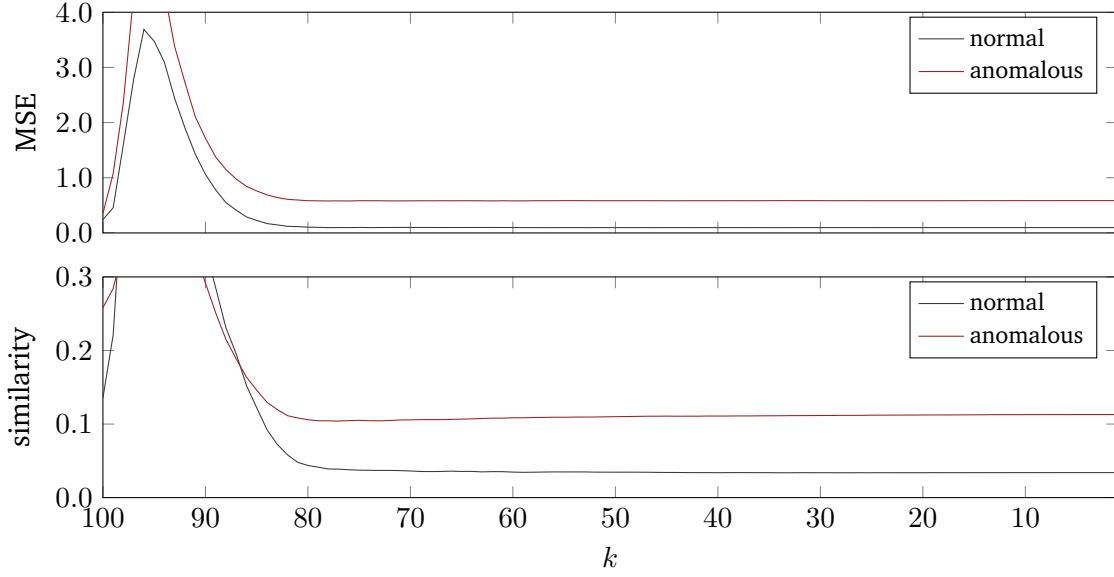


Figure 4.16: Reconstruction error for each diffusion sampling step of arbitrary test samples with and without anomalies.

already demonstrated in [153] that DDPMs can generate comparable results by using a limited number of diffusion steps during sample generation, thereby significantly reducing the inference time of the generation process. Instead of stopping the backward process when the quality of the generated sample stops improving, they skip most of the intermediate diffusion steps. However, in a reconstruction-based scenario, where an easy-to-apply measurement of the reconstruction improvement is available in each step as a reference signal is present, we employ an *early stopping* strategy for the generation, thereby drastically reducing the inference time. Table 4.5 shows the mean generation frequency (Hz) and the mean reconstruction error of normal and anomalous time steps in the complete testing dataset when completing all K steps and when stopping the generation at $k = 70$. With the early stopping applied, we can generate samples approximately three times faster without losing reconstruction accuracy. Replacing the DDPM backward steps with DDIM steps did not result in improved reconstruction performance.

Table 4.5: Effect of an early stopping strategy on sample generation.

	frequency	normal	anomaly
complete sampling	1.03	0.015	0.052
early stopping	3.30	0.017	0.047

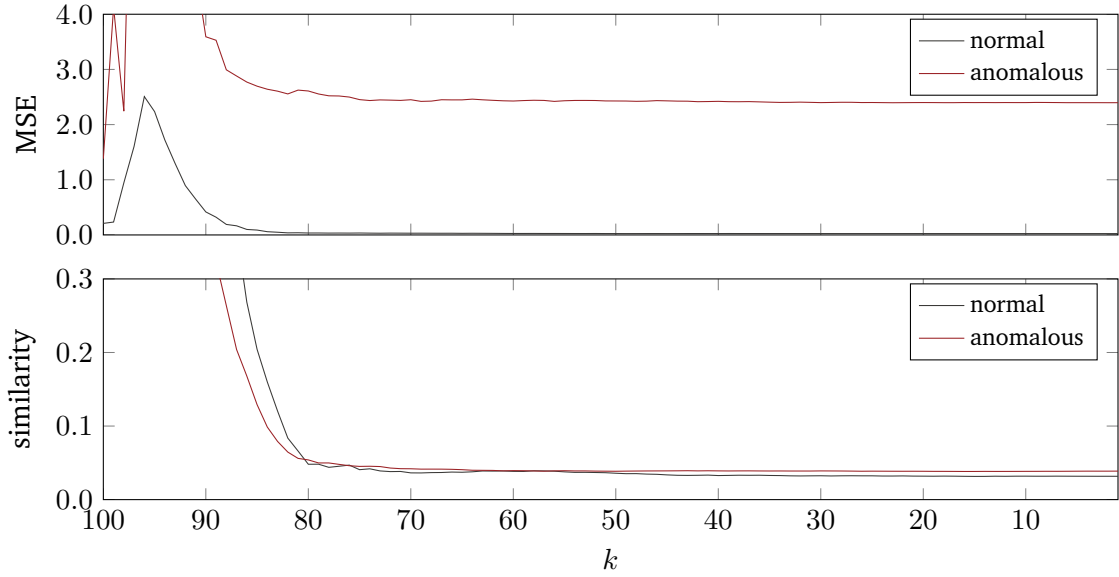


Figure 4.17: Reconstruction of a single channel for each diffusion sampling step of arbitrary test samples with and without anomalies.

4.4.4 Summary

We proposed a diffusion-based model for multivariate time series anomaly detection named *EncoDiff1D*. This model uses the output of a trained encoder to condition a DDPM-based diffusion model for reconstructing multivariate time series data. The encoder is thereby trained prior to the diffusion model for 50 epochs as part of an undercomplete autoencoder, in which the size of the latent representation is adjusted according to the input window size.

The diffusion model is trained on $K = 100$ continuous diffusion steps for 50 epochs using the self-conditioning approach. The loss is computed as a weighted MSE that corresponds to the SNR of each diffusion step. A quadratic scheduler provides the necessary α and β values for each step. As a backbone model, a U-Net with linear attention layers is utilized to predict the noise added during the forward process.

The sample generation begins with Gaussian noise and results in the reconstruction of the input sample, as relevant information is stored in the condition. To accelerate the traditionally slow diffusion process, we truncate the generation process after only 30 iterations. This approach mitigates the main drawback of diffusion models, while preserving reconstruction accuracy.

4.5 Experimental Comparison of the Introduced Models

A critical hyperparameter for detection accuracy and inference timing is the input window size. Each model was trained with the same data configurations (window sizes, strides, and preprocessing steps), training parameters (optimizer and learning rate), and a comparable model structure (layer depth). The number of training epochs varies between models, as GANs typically need longer to converge, especially when only convolutional layers are involved. It is essential to note that an increased input window length reduces the number of iterations per epoch, as the number of training samples is also reduced. The training of a stable Dual-AE-GAN network took 1.5 – 2 hours (1,000 epochs) while the training of the Swin1D-AD models took 6 – 15 minutes (10 epochs). The diffusion model was trained for 50 epochs in 15 – 100 minutes, including training time of the feature extractor.

When evaluating the different models on the corresponding testing dataset, the reconstruction error can be measured separately for normal and abnormal time steps. Table 4.6 presents the reconstruction errors, calculated as the mean across all channels and time steps, for the various input window sizes. This metric is used to illustrate each model's overall reconstruction capabilities, as well as its robustness to anomalies. A higher ratio in these values indicates a more apparent distinction between normal and anomalous time steps, while a lower level of reconstruction error suggests a more accurate reconstruction. The ratio displayed in the table is calculated between the average reconstruction errors of all investigated input window sizes.

While larger input window sizes make it more challenging for the model to accurately reconstruct the input, they also provide enriched contextual information. The reconstruction accuracy of the convolution-based GANs decreases slightly when the input window size is increased. The performance of Swin1D-AD remains at a constant and low level, yielding the best reconstructions among the proposed models for this specific dataset. EncoDiff1D can benefit from the larger context and decrease the reconstruction error. However, it is also evident that with a decreasing reconstruction error of normal time steps, the reconstruction of anomalies improves when attention layers are involved. Swin1D-AD has the highest ratio between normal and anomalous time steps, making it a robust model for anomaly detection.

Table 4.7 compares the associated inference speed (in Hertz) of generated samples per second with varying input window sizes of the model. While the samples have different lengths, they share the same channel size. The convolution-based GANs require more iterations during training to understand complex patterns and relationships within the data, due to their narrow kernel sizes. They are faster by a factor of 2.5 – 4 compared to the transformer-based Swin1D-AD in terms of inference time. The measured inference for the diffusion model EncoDiff1D involves the sampling loop, as discussed before, and the

Table 4.6: Comparison of reconstruction errors of different models with varying input window sizes.

window size	AE-GAN		Sparse-AE-GAN		Dual-AE-GAN		Swin1D-AD		EncoDiff1D	
	normal	anomaly	normal	anomaly	normal	anomaly	normal	anomaly	normal	anomaly
64	0.010	0.050	0.015	0.060	0.010	0.049	0.001	0.015	0.028	0.069
128	0.006	0.063	0.025	0.081	0.008	0.055	0.002	0.016	0.026	0.060
256	0.010	0.044	0.025	0.082	0.011	0.049	0.001	0.015	0.018	0.051
512	0.009	0.072	0.031	0.086	0.013	0.070	0.001	0.015	0.016	0.047
1,024	0.025	0.090	0.025	0.082	0.016	0.071	0.001	0.017	0.017	0.047
avg. ratio	0.012	0.064	0.024	0.078	0.012	0.059	0.001	0.016	0.021	0.055
	5.33		3.25		4.92		16.00		3.24	

creation of the latent representation. The iterative process is also the main reason for its slow generation. A single backward step is measured with around 110 Hz and is thereby faster than a single pass through Swin1D-AD. However, due to the techniques discussed, the inference speed can be increased from around 1 Hz for 100 DDPM backward steps to more than 3 Hz with 30 steps.

Table 4.7: Comparison of inference speed in reconstructed samples per second for each model with varying input window sizes. The values displayed are given in Hertz (Hz).

window size	AE-GAN	Sparse-AE-GAN	Dual-AE-GAN	Swin1D-AD	EncoDiff1D
64	385.97	367.96	240.57	96.32	3.79
128	368.25	369.72	247.58	92.32	3.77
256	366.56	349.50	239.98	92.11	3.52
512	350.12	381.52	234.34	93.18	3.42
1,024	239.14	308.23	226.27	80.34	3.44

The inference speed for a single sample remains almost constant across all input window sizes, with a slight decrease observed when large windows are used. A more informative way to display the benefit and real-world applicability of large window sizes in a high-frequency environment, such as in an industrial setting, can be seen in Table 4.8. The table shows the inference speed in thousand time steps generated per second (Kilohertz).

All models may have real-world use cases, depending on the sensors used, their sampling speed, and the need for an immediate alert. The slowest version of the proposed models and configurations can still reconstruct 230 time steps per second, which is sufficiently fast in certain scenarios.

Model training processes use the AdamW optimizer [130] instead of the traditional Adam [162]. It is the preferred optimizer for most modern models as it decouples the weight decay step from the gradient-based update. Specifically, it applies L_2 regularization directly to the weights following the adaptive updates, rather than incorporating it

Table 4.8: Comparison of inference speed in reconstructed time steps per second for each model with varying input window sizes. The values displayed are given in Kilohertz (kHz).

window size	AE-GAN	Sparse-AE-GAN	Dual-AE-GAN	Swin1D-AD	EncoDiff1D
64	24.70	23.55	15.87	6.08	0.24
128	47.13	47.32	31.49	12.16	0.48
256	93.84	89.47	60.16	23.81	0.90
512	179.26	195.34	110.59	46.08	1.75
1,024	244.88	315.63	209.92	86.02	3.52

within the gradient calculations [163]. The used models and configurations were not further optimized for each input size. Instead, the same settings were used for comparable results.

4.6 Summary

This chapter introduced several novel model architectures for time series anomaly detection. The models are based on GANs, Transformers, and diffusion models.

We improved traditional GANs to reconstruct input samples rather than generate new ones, thereby eliminating well-known challenges such as mode collapse. The proposed models employ an autoencoder in the generator (AE-GAN) and utilize different techniques to keep the latent representation sparse (Sparse-AE-GAN). The sparse representation prevents the generator from being overly confident in reconstructing the inputs. This limitation is beneficial in an anomaly detection setup, as it forces the generator to learn the significant features of the input data and therefore interferes with its capacity to reconstruct anomalies accurately. Another modification involves the introduction of a second, weaker decoder in the generator, named Dual-AE-GAN. Due to its smaller number of weights, it has limited abilities to reconstruct the input. The weaker reconstructions from Sparse-AE-GAN and Dual-AE-GAN promote a more stable training of the discriminator model. All of the mentioned design choices help stabilize the training and find an equilibrium between the generator and the discriminator, thereby eliminating the disadvantages of ordinary GANs.

Swin1D-AD is another novel model for time series anomaly detection. Based on a U-Net structure and combined with Swin transformer layers, it is capable of computing large input windows, unlike vanilla Transformers. This reconstruction-based approach has proven to be a robust choice due to its adaptability when varying input window sizes are required. This is the case when working with limited data or when several models need to run in parallel with the same configurations. The particular windowed attention algorithm fur-

ther ensures rapid convergence during training and low inference time, which is especially useful in real-world scenarios where sensors operate at high sampling rates.

The proposed EncoDiff1D is a DDPM-based generative model that is conditioned on a latent representation of the input sample. This conditioning technique enables the reconstruction of normal time steps, whereas anomalous ones cannot be mapped efficiently into the latent space in the first place. Due to adjustments in the training and sampling process w.r.t. the ordinary process, the time of reconstructing a sample was reduced, enabling performance for real-world scenarios.

The convolution-based GAN architectures proposed in this chapter clearly have the lowest latency, enabling the usage of sensors with high sampling frequencies. Our transformer-based architecture has by far the best reconstruction capabilities and the highest ratio between normal and abnormal time steps in the shown experiment with relatively low latency. The Swin1D-AD model is a fast, accurate, and scalable approach that can be deployed in real-world scenarios. The shown diffusion-based model is relatively slow due to the iterative sampling process and inaccurate when reconstructing signals.

While we measured reconstruction performance in this section using the MSE for normal and anomalous time steps, there are other performance metrics for anomaly detection. The following chapter introduces different approaches for deriving a comprehensive statement from the model output.

5

EVALUATION STRATEGIES

All mentioned models and their variants are trained to return an anomaly-free reconstruction of the input sample. To identify anomalies, an anomaly score a must be calculated. Furthermore, a threshold τ is used to assign a predicted binary label $\hat{y}$ to each data point at time step t .

$$\hat{y}_t = \begin{cases} 0, & \text{if } a_t < \tau \\ 1, & \text{otherwise.} \end{cases} \quad (5.1)$$

The effectiveness of the algorithm can be evaluated by comparing the predicted label $\hat{y}$ with the actual label y . While publicly available datasets typically provide labeled data, real-world data may be unlabeled, complicating the comparison of different algorithms. Nevertheless, there is a need to calculate anomaly scores and establish an appropriate threshold in both scenarios.

In the following sections of this chapter, various algorithms for determining the anomaly score and the threshold are presented in Section 5.1. Furthermore, Section 5.2 introduces various metrics for evaluation, enabling the comparison of different models. Section 5.3 summarizes this chapter.

5.1 Analysis of Anomaly Scoring and Thresholding Techniques

Obtaining meaningful anomaly scores is essential for the efficacy of anomaly detection methods. As explained in Section 2.2.1, the existence of diverse anomaly types necessitates the implementation of distinct scoring algorithms tailored to their characteristics. For point anomalies, it is necessary to derive an individual anomaly score at every time step t . Conversely, collective anomalies necessitate assigning an anomaly score to a subsequence.

Utilizing networks that reconstruct or impute data facilitates a direct comparison between the (unmasked) input and output samples. Given the absence of a standardized procedure for computing the anomaly scores [164], the methodologies discussed in Section 3.2 employ various techniques, typically involving adaptations of their respective loss functions. All referenced approaches described use the reconstruction error at each time step t as a fundamental component in computing the anomaly score. The reconstruction error is thereby usually calculated point-wise through metrics such as Mean Absolute Error (MAE) or MSE for each discrete time step. However, area-wise calculations or evaluations in the latent space may also be considered [104] to supplement the reconstruction error. A possible disadvantage when utilizing point-wise measurements only is visualized in the hypothetical case pictured in Figure 5.1. The input signal was reconstructed accurately, but is shifted by 50 time steps. Even though the reconstruction resembles the signal perfectly, the resulting MSE used as an anomaly score yields high values because the peak is reconstructed at the wrong position. The high scores incorrectly indicate the presence of an anomaly.

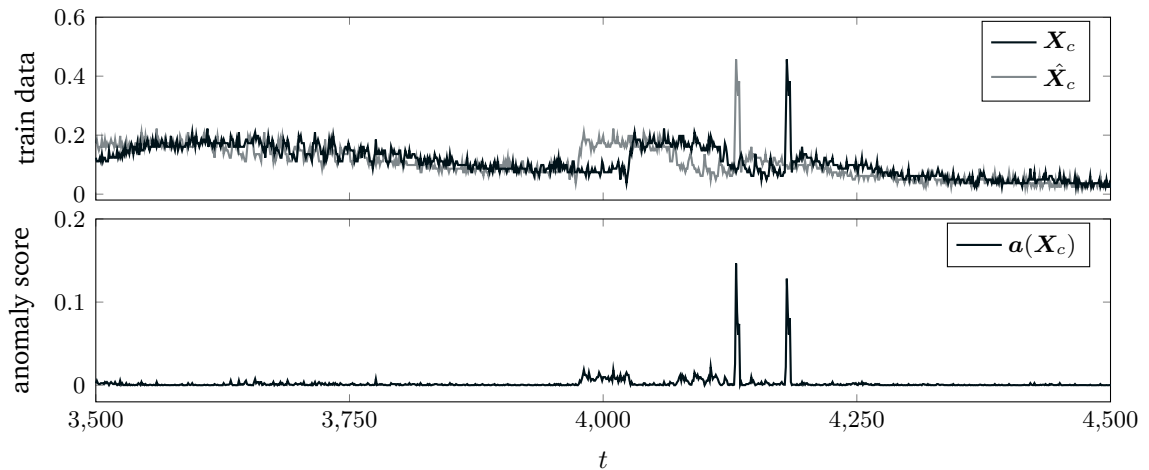


Figure 5.1: Top: A segment from the first channel of entity machine-1-6 of SMD and the shifted (reconstructed) version. Bottom: The corresponding anomaly score calculated as the MSE.

To overcome this drawback, various similarity scores, such as DTW [17], as discussed in Section 2.2.1, are incorporated by approaches like [20]. Moreover, GAN-based methods leverage the output of the discriminator $\mathcal{D}(\mathbf{X})$ as an additional element within the anomaly score computation [20, 98]. The authors of AnomalyTransformer [105] integrate the association discrepancy, which is utilized during training in their attention module and can be seen as a cornerstone of their loss function. The diffusion-based approach ImDiffusion [114] employs a composite of reconstruction errors as its anomaly score. The reconstruction errors are derived from the predictions of various diffusion steps.

When multiple components contribute to the calculation of the anomaly score, they are often aggregated through a convex combination. An overview of the different elements of the anomaly score calculations is presented in Table 5.1.

Table 5.1: Anomaly score calculations of different approaches besides the standard reconstruction error in the time domain.

Model	Anomaly score calculation
MAD-GAN	Discriminator output
TadGAN	Similarity score, discriminator output
USAD	-
TranAD	-
MEMTO	Reconstruction error in the latent space
AnomalyTr.	Association discrepancy
DiffAD	-
DDMT	-
DiffusionAE	-
ImDiffusion	Intermediate reconstruction errors

A problem can occur when using metrics that average the anomaly scores across channels. Suppose the anomaly is only present in one of numerous channels, and the model is perfectly trained to reconstruct the normal time steps, but it successfully fails to rebuild the anomaly. Depending on the extent of the anomaly, the corresponding mean anomaly score could be slightly higher than the scores of the others. Figure 5.2 illustrates a subsequence of a publicly available dataset and its reconstruction (top)¹. The figure shows only one of the 38 channels where the anomaly is visually most noticeable. The anomaly score for the displayed channel and the anomaly score as the mean across all channels are visualized below. In this example, the anomaly score is calculated as the MSE between the input signal and its reconstruction. The anomaly (highlighted area), particularly the large one, was not reconstructed, as it deviates significantly from the remainder. However, when focusing on the corresponding mean anomaly score, this separation is not as clear.

¹The dataset is introduced in the following chapter and is chosen for illustrative purposes only. It is the same time series as used before in Section 4.3.3. As a model, a version of Swin1D-AD is utilized.

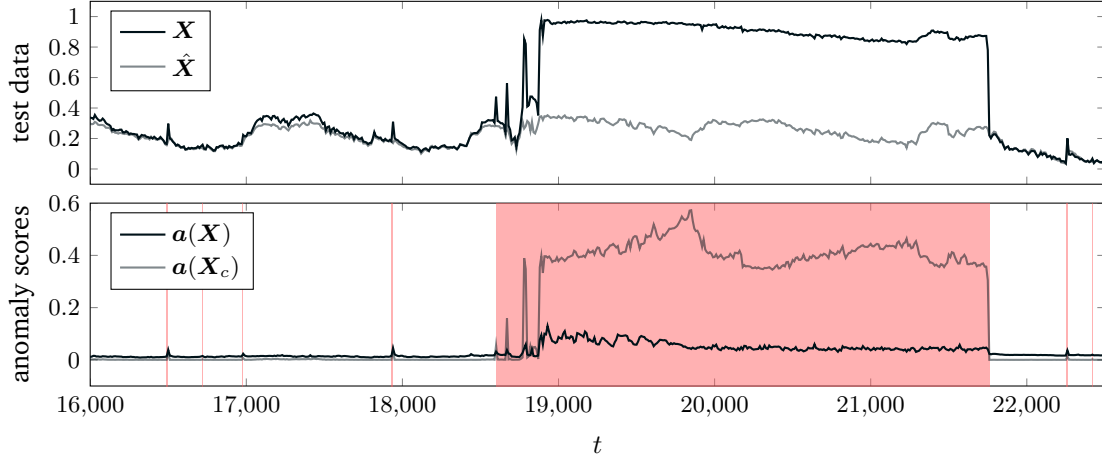


Figure 5.2: Top: A segment from the first channel of entity machine-1-6 of SMD and the reconstructed signal. Bottom: The corresponding anomaly score for the displayed channel, as well as the mean anomaly score across all channels.

The inconsistency in calculating the anomaly score complicates its interpretation and analysis [165]. The anomaly scores of individual time steps can only be compared within the same dataset or domain when the same detection method was used, as they lack a universal meaning. To enhance comparability, regularization and normalization are necessary [164]. By normalizing $a_t \in [0, 1]$, the anomaly score can be seen as a probability of an anomaly being present at time step t . However, a normalization requires statistics about the anomaly scores, such as minimum and maximum values, which are only available when evaluating the testing dataset. Deriving those statistics from the training dataset, which typically does not include anomalous data, would yield anomaly scores that fail to meet the necessary constraints, e.g., the data range.

Most anomaly detection algorithms do not normalize the anomaly score. Instead, they utilize the calculated values and assign a binary label to each time step based on the value relative to a specific threshold. This threshold is often determined using the anomaly scores from either the training dataset [112] or the validation dataset [105, 116]. During inference, an anomaly can be detected immediately, as no further calculations beyond anomaly scoring are required. It is only necessary to check whether the anomaly value is above or below the established threshold.

The following list describes frequently used methods for finding the threshold:

- **POT:** The Peaks Over Threshold (POT) method [166] is used in [92, 103] to determine the threshold. POT is a statistical approach based on the generalized Pareto distribution for finding extreme values in the data. The threshold is typically set high, following the assumption that it identifies rare events. However, outliers in the anomaly score, e.g., when there is an extremely high value that was not re-

constructed, can have a negative impact on determining the threshold and, consequently, on the detection performance.

- **Gap Statistics:** When the anomaly scores are calculated, the threshold is found by setting the top-p% of the validation data anomalous [105, 116] with the gap statistics method. Sometimes the training data is used for this method in addition to or as a replacement for the validation data [104], as validation data is not always available, e.g., when working with a limited amount of data. The proportion is thereby derived from the dataset statistics, which have to be known in the first place.
- **Static Threshold:** In [20], the authors choose to use a sliding window over the anomaly scores. The threshold of a window is set as four times the standard deviation from the mean of the window, similar to what was discussed in Section 2.2.
- **Iterative Search:** A straightforward approach to find a threshold is to test different threshold values, compute anomaly detection metrics that will be discussed in the following section, and choose the value that yields the best detection results [115]. However, a search range and step size must be specified for identifying the best possible threshold. This method is only suitable if the anomaly scores for the labeled test data are available.

Figure 5.3 illustrates the mean anomaly scores across all channels as displayed earlier, along with the four different thresholds as listed above. Again, the labeled anomaly is illustrated by the red areas, which are only visualized in the top section of each graph. The blue lines represent the threshold, derived by the respective algorithm. When the anomaly score falls within the blue area, the time steps are considered normal. Exceeding values indicate anomalies. The figure shows the different outcomes of the algorithms. While POT can detect the first part of the consecutive anomaly, the later part and the short anomalies (small peaks in anomaly scores) remain undetected. The Gap statistics method, with an anomaly ratio of 0.5% for the entities of SMD, as also used in [105, 104], sets the threshold at a lower level and thus fails to detect a single point in the shown sequence. This threshold is derived from the anomaly scores of the training data. The static threshold shows a band of two times its standard deviation around the mean anomaly score of a given window. With this method applied, only parts of the long anomaly are detected. A drawback of this algorithm is the number of parameters to be set. Despite the factor for standard deviations, which defines the width of the band, the window size and step size have to be set. In this example, this window size is set to $\frac{T'}{10}$ and the step size to $\frac{T'}{100}$, where T' is the length of the entity. The iterative search for a threshold yields the best visual separation between the labeled anomalies and the normal subsequences.

Figure 5.4 shows the anomaly scores of the first channel, as also shown before. The consecutive anomaly is more evident and found completely when using POT and Gap statistics. The latter can also correctly identify the narrow spikes as anomalies. The static

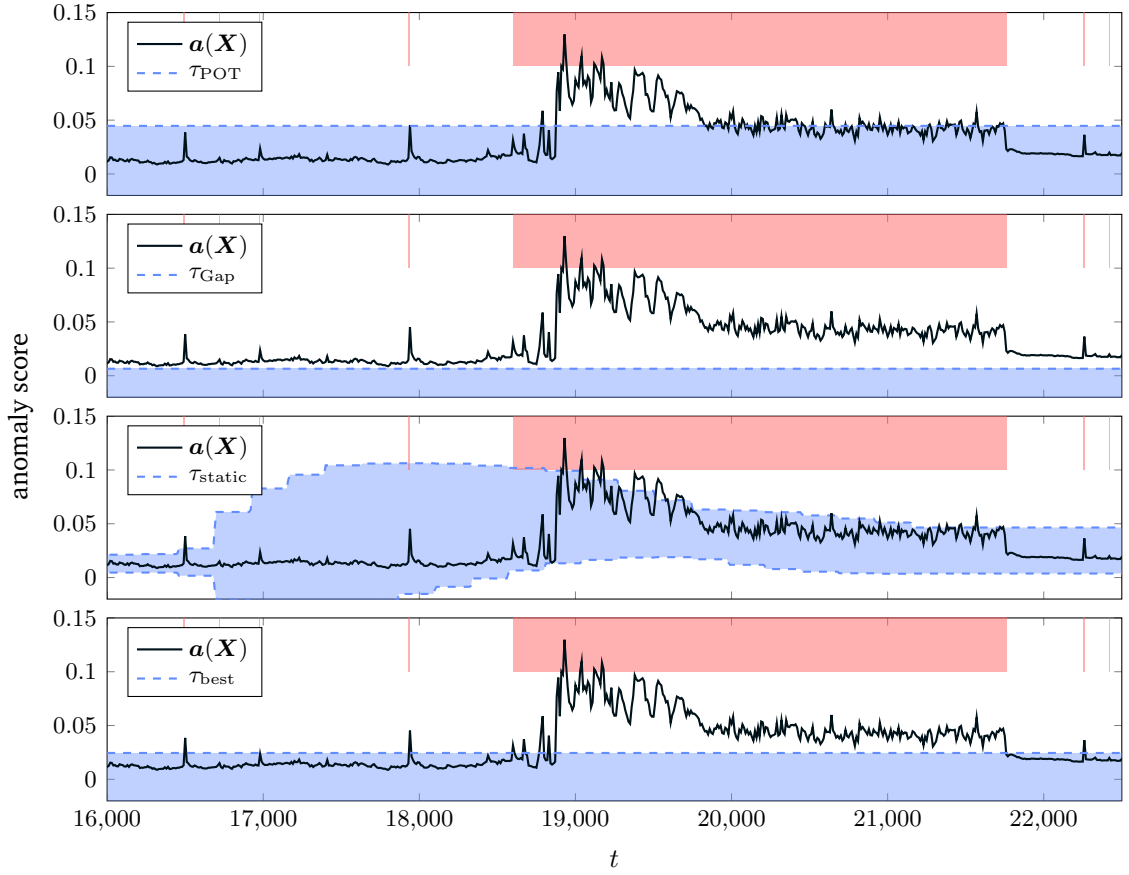


Figure 5.3: Anomaly scores of entity machine-1-6 of SMD along with different thresholds.

threshold fails to detect the entirety of this anomaly, as the higher volatility in the local window results in a larger standard deviation. The iterative search method yields nearly the same threshold as the Gap statistics in this example, therefore yielding similar detection results. As the thresholds are close, the Gap statistics threshold can be considered optimal for these anomaly scores.

5.2 Relevant Performance Metrics

In order to effectively compare the detection performance of different approaches, it is essential to identify meaningful metrics. The F1-score is frequently employed as a point-wise metric for assessing detection accuracy [167]. It is determined by comparing the binary labels (0: no anomaly, 1: anomaly) of the test dataset $\mathbf{y}$ with the predicted labels $\hat{\mathbf{y}}$ derived from the anomaly score. This comparison yields four key values:

1. **TP:** The number of true positives (TPs) reflects the anomalies that were correctly identified, occurring when $\mathbf{y}_t = 1$ and $\hat{\mathbf{y}}_t = 1$.

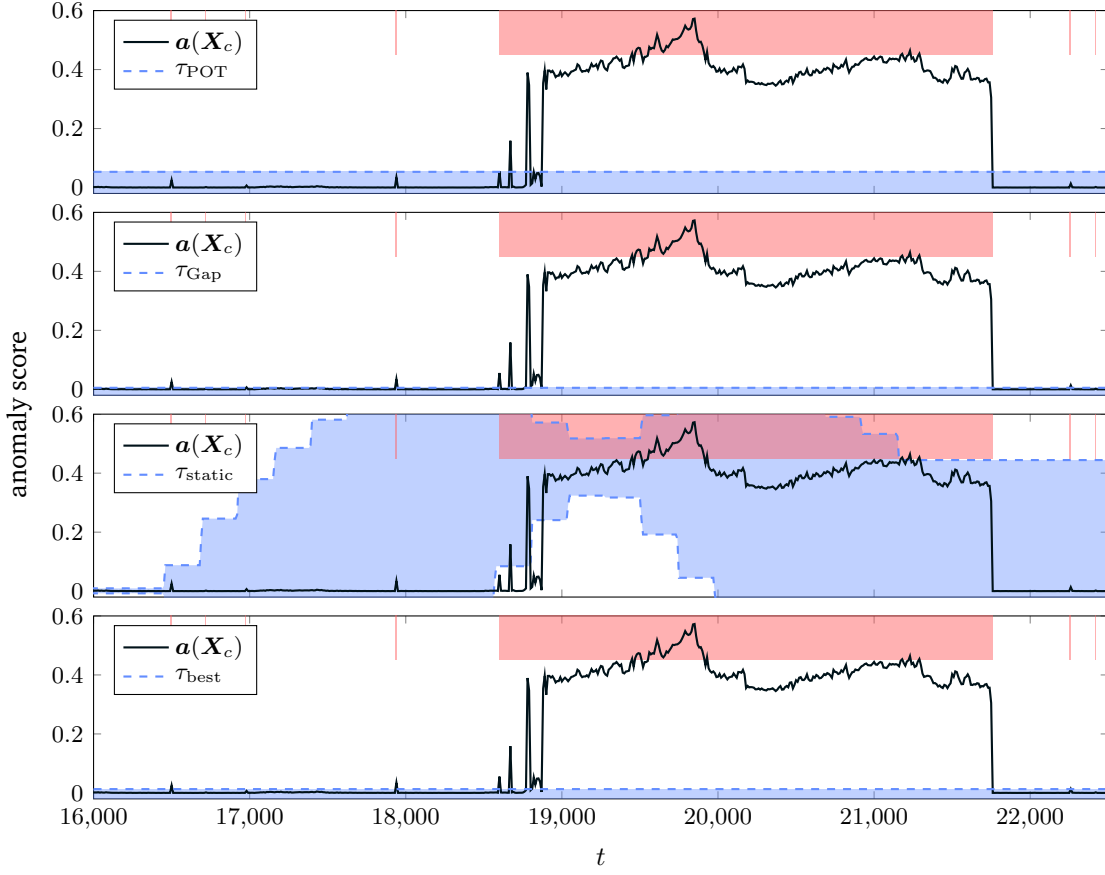


Figure 5.4: Anomaly scores of the first channel of entity machine-1-6 of SMD along with different thresholds.

2. **TN:** The true negatives (TNs) indicate the number of time steps that were correctly classified as normal, represented by $y_t = 0$ and $\hat{y}_t = 0$.
3. **FP:** A false positive (FP) is an error that arises when the anomaly score incorrectly indicates the presence of an anomaly, occurring when $y_t = 0$ and $\hat{y}_t = 1$.
4. **FN:** A false negative (FN) occurs when an anomaly exists at time step t ($y_t = 1$) but is not detected ($\hat{y}_t = 0$).

Those values can be utilized to calculate Precision (P), Recall (R) and their harmonic mean as the F1-score as

$$P = \frac{TP}{TP + FP}, \quad (5.2)$$

$$R = \frac{TP}{TP + FN}, \quad (5.3)$$

$$F1 = 2 \frac{P R}{P + R}, \quad (5.4)$$

5 Evaluation Strategies

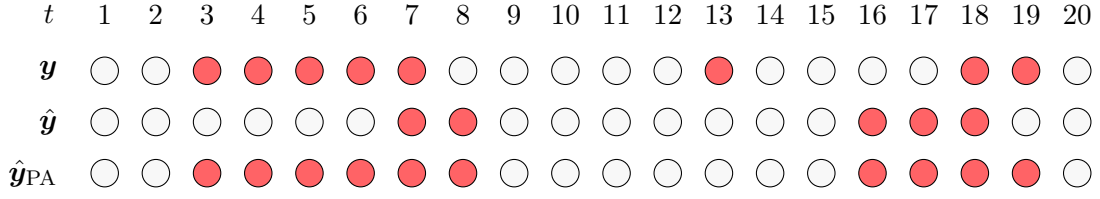


Figure 5.5: Visualization of the point adjustment strategy.

Table 5.2: Example calculation of different metrics.

	TPs	TNs	FPs	FNs	P	R	F1
standard	2	9	3	6	0.40	0.25	0.31
point adjustment	7	9	3	1	0.70	0.88	0.78

where $P, R, F1 \in [0, 1]$. In order to maximize P , R , and therefore the F1-score, the number of TPs has to be high compared to the number of FPs and FNs.

A widely utilized method based on the F1-score is the *point adjustment strategy* [92, 103, 105, 112, 116], henceforth referred to as $F1_{PA}$. This strategy, introduced in [168], operates under the premise that a consecutive anomaly is identified when at least one time step is labeled correctly within that anomaly. An illustration of point adjustment is presented in Figure 5.5, which showcases arbitrary labels y and predicted labels $\hat{y}$ across 20 time steps. The labels resulting from the application of the point adjustment strategy are denoted as $\hat{y}_{PA}$. The highlighted time steps indicate the presence of an anomaly and the corresponding predicted anomaly, respectively. The results for this simplified example are summarized in Table 5.2. By employing the point adjustment strategy, the count of TPs increases, while the count of FNs decreases. Consequently, the F1-score improves from 0.31 with the standard calculation to 0.78. This procedure can be useful in real-world scenarios, as it prioritizes alerting on detected anomalies over precise localization [105].

As the point adjustment strategy can yield overly optimistic results, an alternative protocol, denoted as $PA\%K$, was introduced by Kim et al. [169] where the strategy was computed for various $K \in [0, 100]$ values. To be precise, all predictions in an anomalous segment are considered detected when at least $K\%$ of the points in the segment are correctly classified. The resulting $F1_{PA\%K}$ is the area under the curve for all K . $F1_{PA\%0}$ is thereby equal to $F1_{PA}$ and $F1_{PA\%100}$ is equal to $F1$. The above example would result in $F1_{PA\%K} = 0.44$.

While the F1-scores are commonly used metrics for time series anomaly detection, additional metrics are often employed to highlight the classification performance of an algorithm.

The Receiver Operating Characteristic (ROC) curve, specifically the Area under the ROC Curve (AUROC), is frequently utilized as an additional illustration to describe the performance of a binary classification model, as in [103, 105]. This curve visualizes the true positive rate (TPR) against false positive rate (FPR) for different thresholds, denoted as τ . The TPR can be interpreted as the sensitivity of the algorithm, which is equivalent to the recall R as in Equation 5.3. The values for the ROC curve at each threshold can be calculated as

$$\text{TPR}_\tau = \frac{\text{TP}_\tau}{\text{TP}_\tau + \text{FN}_\tau} , \quad (5.5)$$

$$\text{FPR}_\tau = \frac{\text{FP}_\tau}{\text{FP}_\tau + \text{TN}_\tau} . \quad (5.6)$$

While the ROC curve is widely utilized, a more informative metric is the AUPRC [170], which describes the area under the curve when P against R for various thresholds are plotted. This metric is particularly suitable for evaluating models on imbalanced datasets [171, 172], which are common in anomaly detection scenarios, as it reflects how well a model differentiates the positive class.

Both AUROC and AUPRC are threshold-independent metrics that provide an overview of the performance of a model. However, for the deployment of a detection algorithm in real-world applications, a threshold is essential for triggering alerts.

5.3 Summary

The different thresholding algorithms have their benefits and drawbacks. POT is an elegant method for determining a threshold on the training or validation data. However, a single outlier in the anomaly score can yield a high threshold, resulting in bad overall performance during testing. The gap statistics method requires prior knowledge about the data. In practice, the ratio of anomalous time steps is set to 0.1% – 1% [105], even if the ratio in the publicly available dataset is much higher (later discussed in Section 6.1). The static threshold requires additional calculations during inference as it is dynamically adjusted depending on the anomaly scores. An iterative search for the threshold may yield the best possible results, but it is time-consuming as all calculations for the different metrics must be computed multiple times. It also requires the anomaly scores of the test data to determine the optimal threshold, making it impractical in real-world scenarios.

The listed metrics, especially F1-score with the point adjustment strategy, are the most popular ones, even if a variety of other metrics exist [16]. However, metrics such as volume-based ROC [173] hardly find resonance in recent publications.

When the point adjustment protocol is applied, the resulting score is mainly denoted as F1-score [169] without explicitly mentioning the protocol in the results. This strategy is

frequently criticized [169, 174, 175], as it is unreasonable, and even a random guess can yield good results [176]. Moreover, it is challenging to compare algorithms as different detection results can yield the same $F1_{PA}$, especially when long consecutive anomalies are present in the dataset. For instance, when revisiting the displayed subsequence in Figure 5.2 and the different thresholding algorithms, all identify at least one time step of the large anomaly, thus marking the whole anomaly as detected.

Choi et al. [167] suggest focusing more on the recall in real-world scenarios, as it is crucial for the system to identify as many anomalies as possible, even if this results in false alarms. However, the recall alone is no indicator of a good detection algorithm, as $\hat{y} = 1$ would easily maximize recall, as every anomaly is found, but also results in a significant value for FPs.

The advantages of AUPRC and AUROC are their threshold independence, which eliminates the need for additional algorithms and comparisons. However, in real-world scenarios, a threshold is still required to trigger an alert.

In the following Chapter 6, different algorithms are compared on publicly available datasets incorporating various of the introduced metrics. The real-world data is unlabeled; thus, metrics for detection performance are not practical. The experiments with real-world data are presented in Chapter 7.

6

CASE STUDY: PUBLICLY AVAILABLE DATASETS

In the first case study, frequently used benchmark datasets to evaluate the different models are presented. There are a lot of multivariate time series datasets for anomaly detection published, as listed in [16, 72], but only a few of them are commonly considered in recent publications [99, 103, 104, 112, 114, 115, 116].

Section 6.1 introduces those widely used publicly available datasets and provides a comprehensive analysis of them. The novel models developed in Chapter 4 were trained on datasets with varying hyperparameter settings and underwent multiple ablation studies. We further compare our models with SOTA approaches under comparable circumstances. Those experiments and results are presented in Section 6.2. Section 6.3 summarizes and concludes this chapter.

6.1 Data Analysis

Before training the models and comparing the outcomes of various approaches, it is essential to discuss the datasets thoroughly. Table 6.1 presents an overview of the multivariate datasets and summarizes their key characteristics. The color codes are used throughout this chapter to keep the different datasets easily apart. Soil Moisture Active Passive (SMAP), Mars Science Laboratory rover (MSL), and SMD are multi-entity datasets, each of varying length. Each entity corresponds to a single continuous time series with multiple channels and can be viewed as a single physical unit or asset, such as a single machine. The entities of a dataset should be treated individually, as each can exhibit distinct behavior and anomaly patterns [92, 170, 177]. An individual treatment, therefore, requires a separate set of weights for each entity and model. SMAP and MSL contain data related to a spacecraft, whereas in SMD and Pooled Server Metrics (PSM) server metrics are recorded. Secure Water Treatment (SWaT) and Water Distribution (WADI) provide data gathered from water treatment and distribution systems. By default, each dataset is split into training and testing subsets. The minimum, maximum, and average lengths of the entities are detailed in Table 6.2. More comprehensive tables of each entity can be found in Appendix A.1. It is assumed that the training data only reflects the nominal state [176]. The testing data is labeled and includes anomalous time steps. The percentage of anomalies in the table refers to the testing data, as such irregularities do not occur in the training sets.

Table 6.1: Overview of benchmark datasets for time series anomaly detection datasets.

Dataset	Entities	Channels	Domain
SMAP	54	25	Space
MSL	27	55	Space
SMD	28	38	Server
PSM	1	25	Server
SWaT	1	51	Water
WADI	1	127	Water

Each multivariate time series $\mathcal{X} = \{\mathbf{x}_t \in \mathbb{R}^c \mid t = 1, 2, \dots, T'\}$ with T' time steps and c channels at every time step t is split into windows of equal length T . Sample i can thus be written as

$$\mathbf{X}^{(i)} = \{\mathbf{x}_t, \mathbf{x}_{t+1}, \dots, \mathbf{x}_{t+T}\} \quad \text{for } t = (i-1)\Delta + 1, \quad i = 1, 2, \dots, N, \quad (6.1)$$

where $N = \lfloor (T' - T) / \Delta \rfloor + 1$ is the number of extracted samples from the time series that form a dataset for machine learning approaches, and $\Delta > 0$ is the stride between neighboring samples. $\Delta < T$ results in overlapping samples. Using $\Delta = 1$ would maxi-

Table 6.2: Number of time steps for training and testing data of different datasets along with the anomaly ratio.

Dataset	Training Time Steps			Testing Time Steps			% Anomalies
	min.	max.	avg.	min.	max.	avg.	
SMAP	312	2,881	2,555	4,453	8,640	8,070	12.81
MSL	439	4,308	2,160	1,096	6,100	2,730	10.53
SMD	23,687	28,743	25,300	23,687	28,743	25,300	4.21
PSM			132,481			87,841	27.76
SWaT			49,500			44,991	12.20
WADI			118,800			17,280	5.77

mize the number of samples but reduce the variability of the dataset as samples are highly similar and thus cause overfitting. Adjacent samples are highly correlated or even redundant, which can lead to overfitting. Mixing samples efficiently can reduce intra-batch similarities, but the possibilities are still limited [178].

For the remainder of this section, all the datasets are discussed. Some of the following illustrations were created based on the work of Wagner et al. [170]. The datasets are grouped by domain (see Table 6.1). SMAP and MSL were released in the same publication, so the analysis is done together. The same applies to SWaT and WADI. While both SMD and PSM are related to server monitoring, they are structured differently.

In some of the following figures, training data (dark) and test data (bright) are displayed side by side. It is essential to note that the test data presented in those figures represents only the nominal state data. Time steps with anomalies are excluded from these comparisons.

SMAP and MSL [177].

As part of the 2015 Earth Systematic Mission program of the National Aeronautics and Space Administration (NASA), measurements from the SMAP mission were collected. The SMAP satellite was equipped with instruments to measure soil moisture content and related geophysical parameters across the surface of the Earth. This data includes instrument-related metrics, such as sensor status and raw signal measurements, as well as geophysical variables like soil moisture, temperature, and vegetation indices [179]. The MSL dataset was made public along with SMAP and also represents spacecraft-related data. In 2012, the MSL mission landed the Curiosity rover on Mars as part of the Mars Exploration Program. The MSL rover is instrumented with various devices for analyzing samples and monitoring atmospheric conditions, surface composition, and radiation levels [180]. Reports from the Incident Surprise Anomaly (ISA) were used to label unusual events within the SMAP and MSL data. These reports document off-nominal incidents or

operational anomalies observed during the mission, including those caused by space environment interactions or human-induced errors [177]. However, for a time series anomaly detection setup, only a selected number of sensors are provided for both datasets.

The SMAP dataset has $c = 25$ channels at each time step and includes 54 entities. A detailed examination of the raw time series reveals that all channels, except for the first, exhibit a binary nature. The first channel of each entity shows a continuous signal within the range $[-1.00, 1.00]$. While these range limits are reached for every test time series of the dataset, the data range of the training data may be narrower. For instance, the range of the training data for entity D-1 is $[-1.00, -0.58]$, and for entity A-1, the value of the first channel remains constant throughout the entire recording ($x_{t,0} = 0.999$). When comparing the training and testing data of a single entity, noticeable discrepancies are visible. Figure 6.1 visualizes the first channel of entity P-3 for training data (top) and testing data (bottom) on the left, alongside a horizontal histogram displaying the distribution of the values on the right. The highlighted region in the testing time series indicates the labeled anomaly.

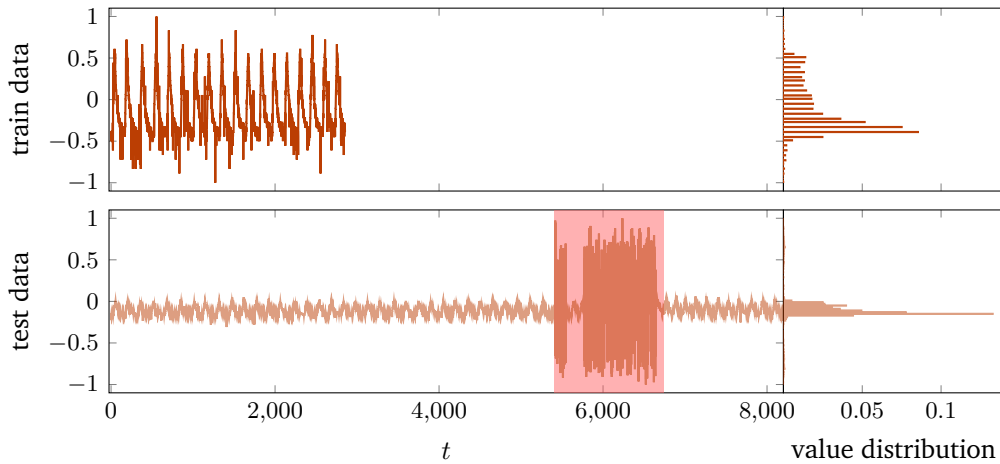


Figure 6.1: The first channel of entity P-3 of the SMAP dataset (left) with the corresponding value distribution (right). The top row visualizes the training data and the bottom row the testing data, where the labeled anomaly is highlighted.

The shared x-axis displays the different lengths (cf. Table 6.2) of both time series, with values scaled to fall within the range of $[-1.00, 1.00]$. The figure further illustrates the discrepancies between the training and the test datasets, visualized by the histograms. While the training data is more uniformly distributed across the entire value range, displaying a denser concentration around -0.4 in the histogram, the testing data is primarily concentrated between -0.2 and 0 . The highlighted anomaly stands out as a high-frequency pattern with extreme values relative to the remainder of the test series.

Figure 6.2 provides a smaller subsequence of the aforementioned signal. The similarity of the patterns is evident when using different scales on the y-axis. The discrepancy in the value range can be challenging when following the assumption that we aim to train an anomaly detection model on the training data to learn the nominal representation of the data, given that the nominal data in the test dataset differs in nature. Although the displayed time series is used as an example of SMAP, this scaling phenomenon does not occur uniformly across all entities, which makes it challenging to identify a universally valid approach.

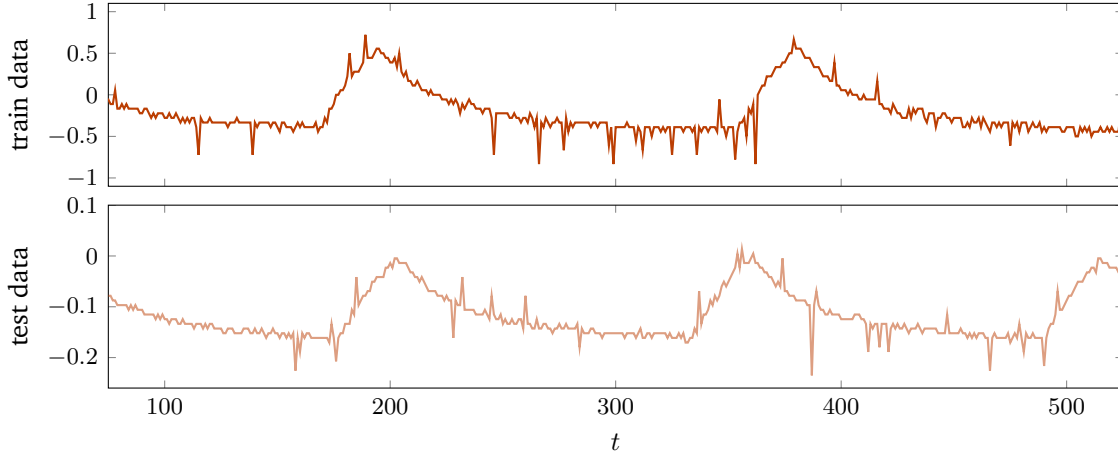


Figure 6.2: A subsequence of the first channel of entity P-3 of the SMAP dataset.

The MSL dataset is highly connected to SMAP due to domain and publisher. The dataset consists of 27 entities, each with $c = 55$ channels. The majority of the channels are binary encoded, with the first channel representing a continuous signal. The value range of half the entities also falls within the range of $[-1.00, 1.00]$, while the other half exceeds this boundary. Some entities exhibit the same behavior of scale mismatches as described before, while others show different nominal patterns during training and testing.

Further focusing on channel 0, the data distributions of different entities of both datasets are shown in Figures 6.3 and 6.4. The box plots reveal the aforementioned discrepancies in data distribution within an entity and differences in characteristics across entities of the same dataset.

An analysis of feature correlations, using Pearson's method [181], revealed that there are hardly connections between the first channel and the binary signals in both datasets. Temporal correlations, such as periodicity or repeating patterns within a channel, can be measured with the autocorrelation function, defined as

$$\rho_l(x_t) = \frac{\text{Cov}(x_t, x_{t-l})}{\sqrt{\text{Var}(x_t) \text{Var}(x_{t-l})}} \quad (6.2)$$

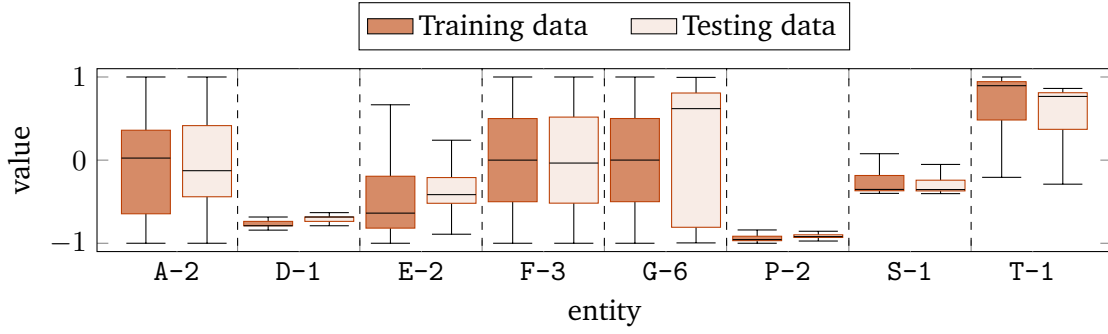


Figure 6.3: Nominal data distribution for channel 0 of different entities of the SMAP dataset.

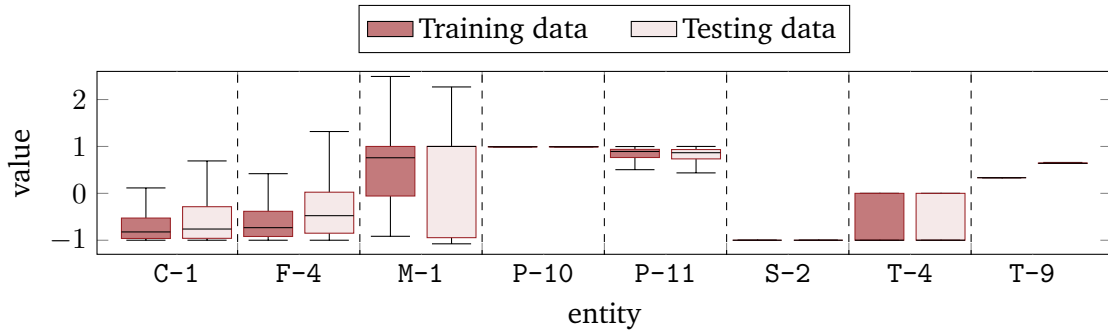


Figure 6.4: Nominal data distribution for channel 0 of different entities of the MSL dataset.

for every lag $l < T$ and $t \geq l$. Autocorrelations of the first channels of two training dataset entities are visualized in Figure 6.5 for $l = 1000$ lags along with their 95% confidence interval. The autoregressive character of entity P-3 of SMAP (the same entity as discussed before) is clearly visible in Figure 6.5a. The autocorrelation function of entity M-2 of the MSL dataset converges quickly towards zero, as can be seen in Figure 6.5b. However, for distant lags the signal has a higher autocorrelation value ($l = 800$).

Another key aspect of anomaly detection datasets is the presence of labeled anomalies in the testing data. While the ratio of anomalous points for each dataset is detailed in Table 6.1, an in-depth examination of the individual anomalies can reveal additional information about the dataset as a whole and the nature of the anomalies. Each entity of SMAP and MSL has one to three labeled anomalous segments of different lengths. The length of each anomaly can be seen in the histograms in Figures 6.6a and 6.7a. For SMAP, most anomalies are shorter than 200, with a few unusually long segments. In the MSL entities, individual longer anomalous segments of more than 1,000 points are present. Given the limited number of data points within the entities, long anomalous segments seem unlikely.

Figures 6.6b and 6.7b show the relative position of the anomalous data points in both datasets across all entities. Especially the anomalies within the entities of SMAP show a concentration in the second half of the time series.

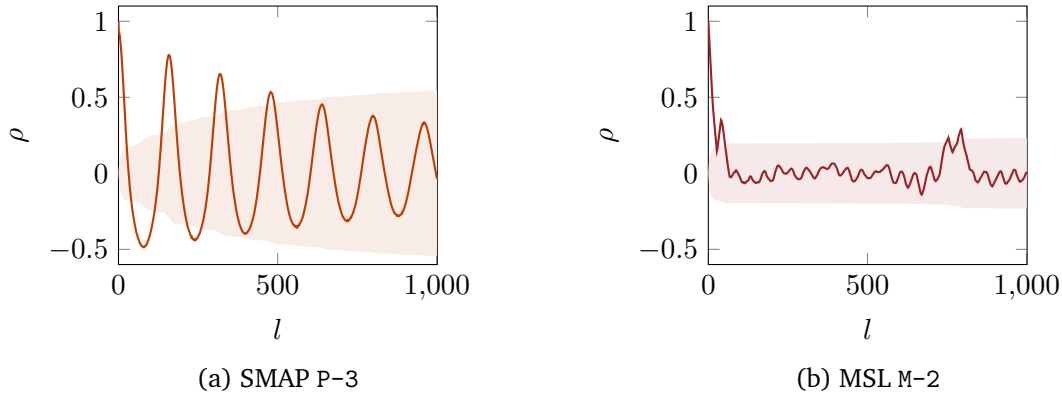


Figure 6.5: Autocorrelation function with 95% confidence band. Only the first channel is analyzed.

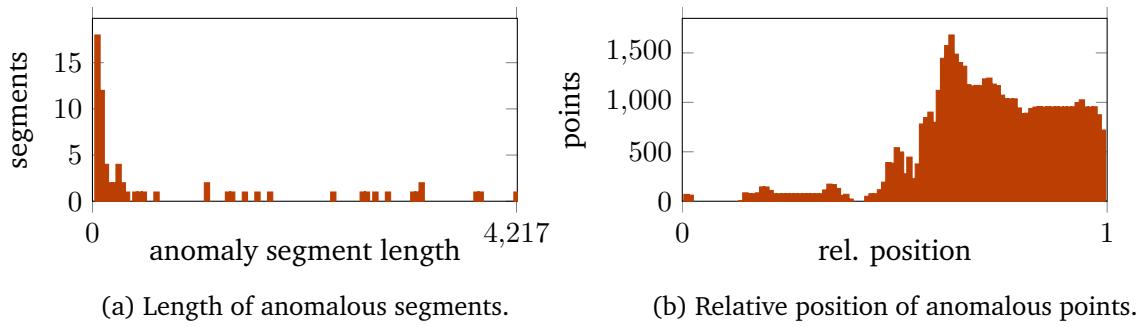


Figure 6.6: Analysis of the anomalies in the SMAP test dataset across all entities.

SMD [92].

The 28 entities of the SMD dataset were collected by a large internet company over a five-week period and made publicly available in 2019. The company monitored server-related metrics, such as network and memory usage, as well as the utilization of various components, and ended up with 38 channels per time step. Based on incident reports, domain experts manually labeled significant time steps in the second half of the recorded data, which is used for testing. All entities are of similar size and range from 23,687 to 28,743 time steps. As Figure A.1a exemplifies for entity *machine-1-1*, the data has been normalized to $[0, 1]$ with some channels remaining constant. A more detailed analysis of the data distributions reveals a notable difference for specific channels when comparing training and test data, particularly for channels four and five, as shown in Figure 6.8. Contrary to SMAP and MSL, SMD does not include channels with binary signals.

Further discrepancies between different entities are visualized in Figure 6.9, where the spatial relationships of the training data associated with the entities *machine-1-1* and *machine-1-6* of SMD are illustrated. While the majority of channels in both entities exhibit a strong positive correlation, differences become apparent, even though the same system

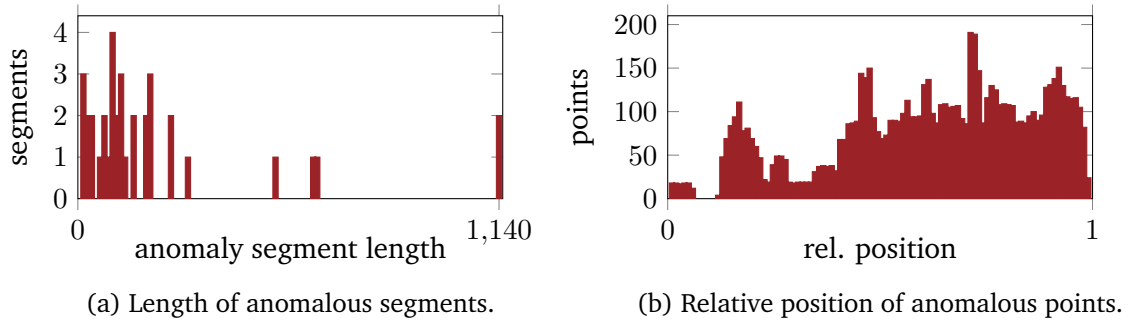


Figure 6.7: Analysis of the anomalies in the MSL test dataset across all entities.

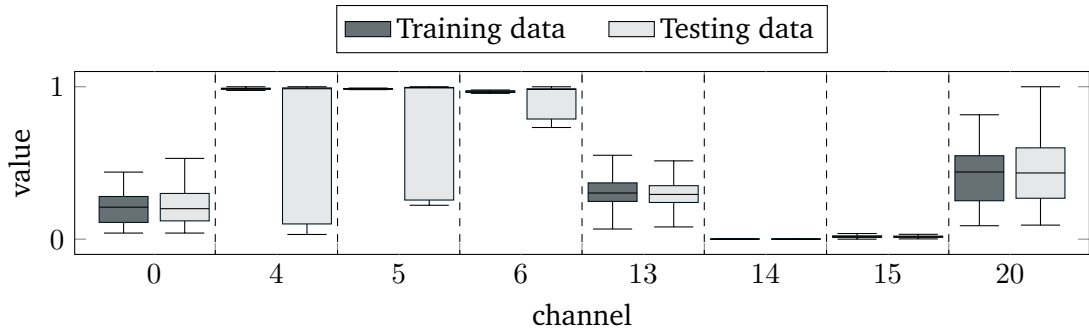


Figure 6.8: Nominal data distribution for selected channels of entity machine-3-11 of the SMD dataset

was observed. A striking difference is evident in the correlations with channel 4 to 10 with 15 to 37, where hardly correlated values are visible in machine-1-6.

In some entities of the dataset, anomalies encounter long-term effects on the signal, e.g., recovery phases, that are not labeled and should thus represent the normal state. This imprecise labeling is often criticized because the number of FP increases with the length of the long-term effect, which negatively impacts the resulting F1 score. Wagner et al. [170] excluded 13 entities of the dataset due to this inaccuracy. One example of those is machine-3-11, pictured in Figure ??, where the signal does not recover from a sudden drop. Its feature correlation matrix is also far from the ones illustrated before.

Each SMD entity has between 1 and 30 anomalous segments. The relative position of anomalies in the testing data is distributed more evenly than that of SMAP, with a slight increase towards the second half of the time series, according to Figure 6.10b. Figure 6.10a shows the lengths of the anomalous segments across all entities, where most anomalies are shorter than 50 time steps. It is crucial to note that the plots in Figure 6.10 display the accumulated data across all entities, as done previously for SMAP and MSL.

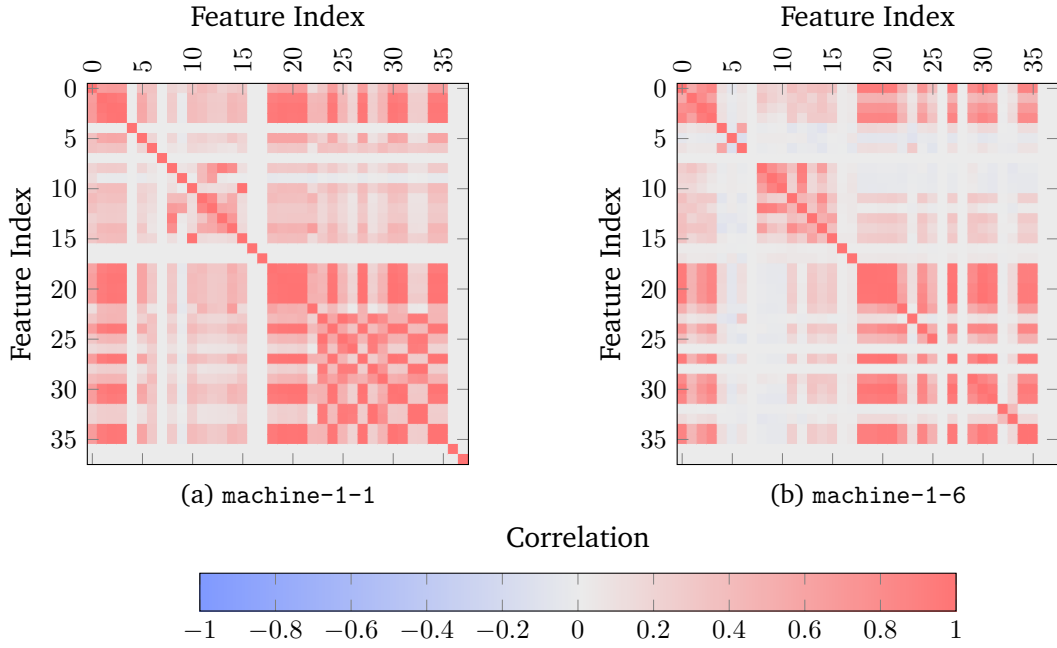


Figure 6.9: Feature correlation of different entities from the SMD training dataset.

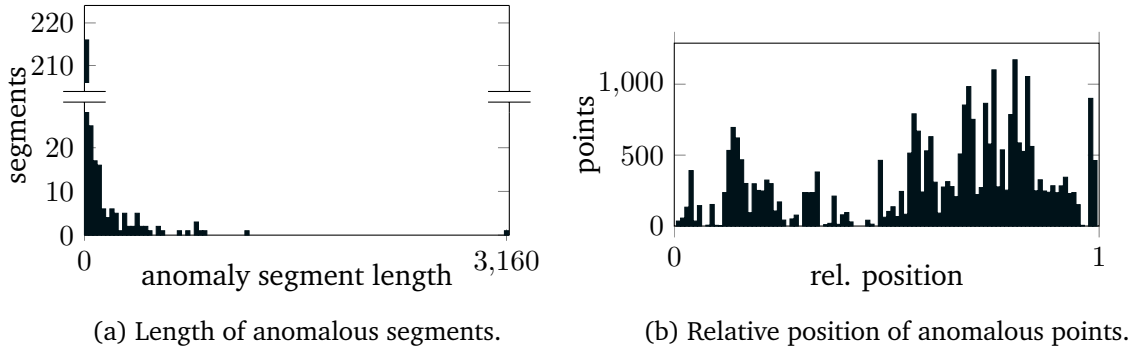


Figure 6.10: Analysis of the anomalies in the SMD test dataset across all entities.

PSM [182].

The PSM dataset is another widely used server dataset, released in 2021, making it the latest of the datasets presented. The data was collected in a single entity from multiple application server nodes at *eBay* and represents several relevant server machine metrics, similar to SMD. With 132,481 time steps, it is the longest training entity within the used datasets. Figure A.1b shows a value range of $[0, 1]$ for all the channels without any constant values. However, the data range for training and testing may differ drastically when compared side by side. The most noticeable is channel 23. Even when exploring the corresponding box plot in Figure 6.11, the dissimilarities between the training and testing data are evident for this channel.

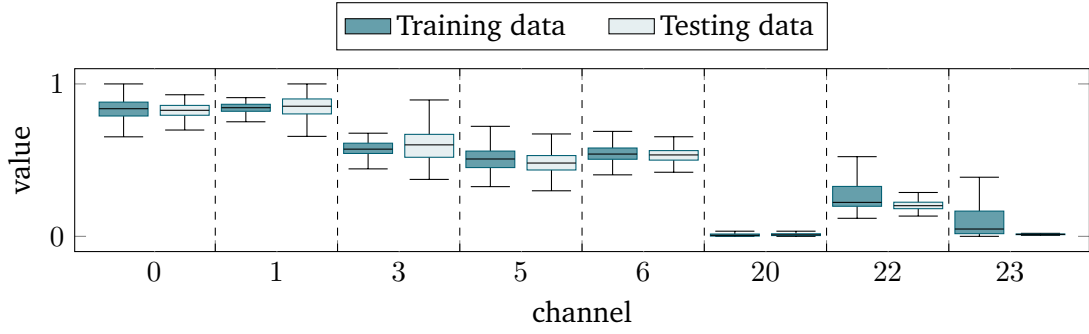
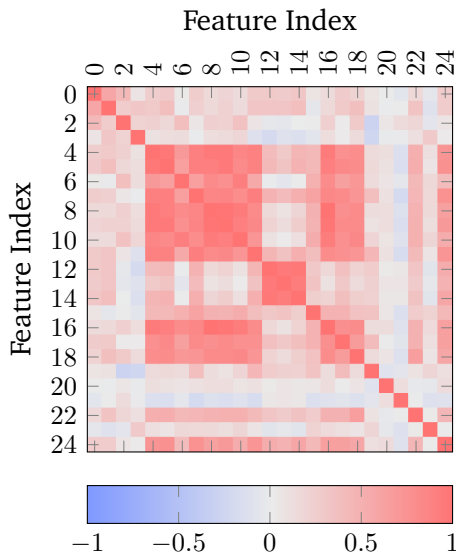
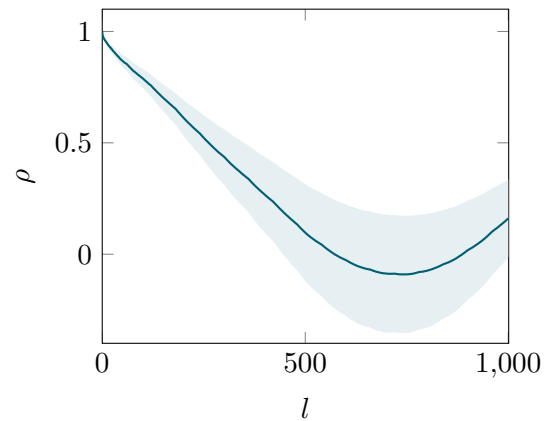


Figure 6.11: Nominal data distribution for selected channels of the PSM dataset.

Figure 6.12a visualizes the feature correlations within the training data of the PSM dataset. As the dataset is published anonymized, there is no mapping between channels and sensors. Still, three main clusters are visible: channels four to eleven are highly connected among themselves and with channels 16 to 18. The correlated channels appear to be the outputs of the same sensor, as their visual appearance, data ranges, and value distributions are similar. Additionally, their autocorrelation functions exhibit a similar shape. Figure 6.12b illustrates the mean autocorrelation graph across the mentioned channels with the corresponding standard deviation. Another small cluster can be seen at channels 12 to 14. Again, channel 23, along with a few others, shows no outstanding correlations with other channels of the training data.



(a) Feature correlation of the PSM training dataset.



(b) Mean autocorrelation function across feature-correlated channels with the corresponding standard deviation.

Figure 6.12: Feature and time correlation of the PSM training dataset.

Anomalous segments were labeled manually by experts [182]. Most of the 72 anomalous segments are shorter than 50 time steps (see Figure 6.13a), with a few longer exceptions. The longest continuous anomaly spans 8,860 time steps, which alone resembles approximately 10% of the testing data. The long anomalies form the clusters of anomalies that can be seen in Figure 6.13b.

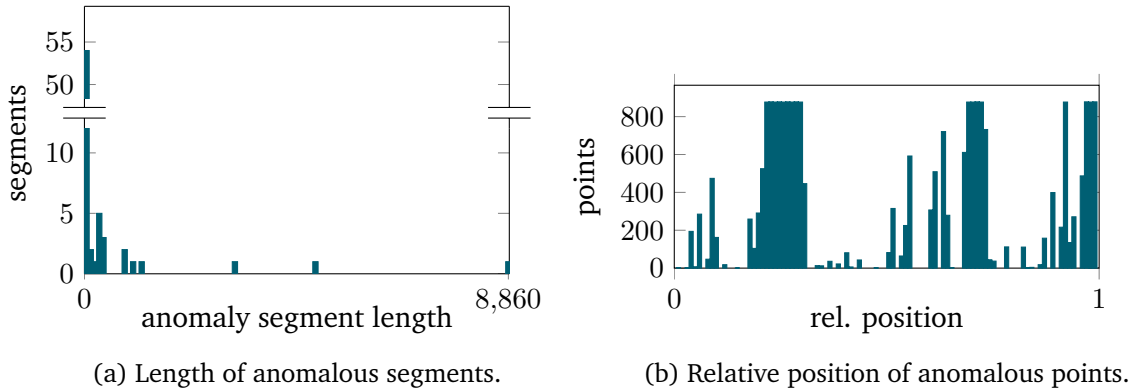


Figure 6.13: Analysis of the anomalies in the PSM test dataset.

SWaT [183, 184] and WADI [185].

The research group of SWaT and WADI recorded data from a water-treatment and water-distribution testbed, respectively. During data collection over eleven and 16 days from several sensors, actuators, and network-related devices, simulated cyber and physical attacks were injected, resulting in anomalies. The testbed scenario and simulated attacks also resulted in accurate labeling. The data from SWaT and WADI include measurements from 51 and 127 channels, respectively. The values of the training data are in the $[0, 1]$ range, which is not necessarily the case with the test data (as can be seen in Figures A.1c and A.1d in the appendix). When taking the box plots of the channels into account that exceed the data range, e.g., channel 1 of WADI, the data distribution of training and testing data looks similar (Figure 6.15), indicating outliers to be present and to distort the range plots. However, this is not always the case, as channel 4 of SWaT shows. The data box plot shows significant differences in some of the visualized channels (Figure 6.14). Figure 6.16 illustrates the correlations among features within the training data of the SWaT dataset. There exist strong positive and negative correlations, particularly between channels 25 and 45.

Most of the anomalous segments of the SWaT dataset are short with less than 100 time steps (Figure 6.17a). A single, unusually long anomaly occurs near the center of the dataset, with more than 3,500 continuous time steps being labeled as anomalous. This anomaly forms the cluster in Figure 6.17b. The other anomalies are present at the beginning and end of the dataset. The test data of the WADI dataset is relatively short,

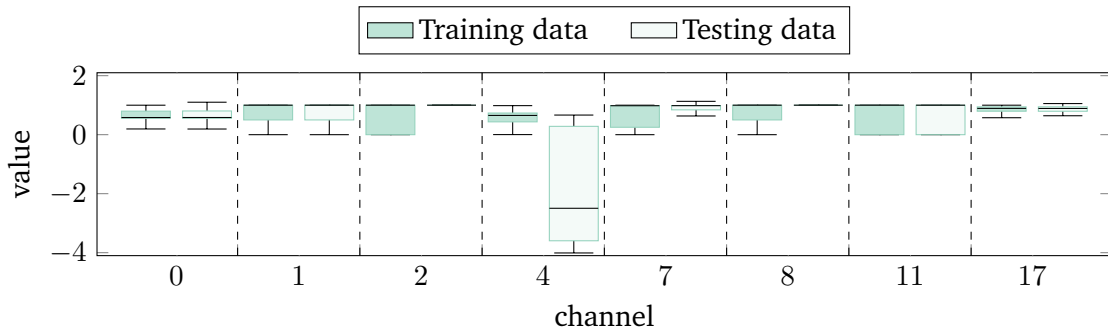


Figure 6.14: Nominal data distribution for selected channels of the SWaT dataset.

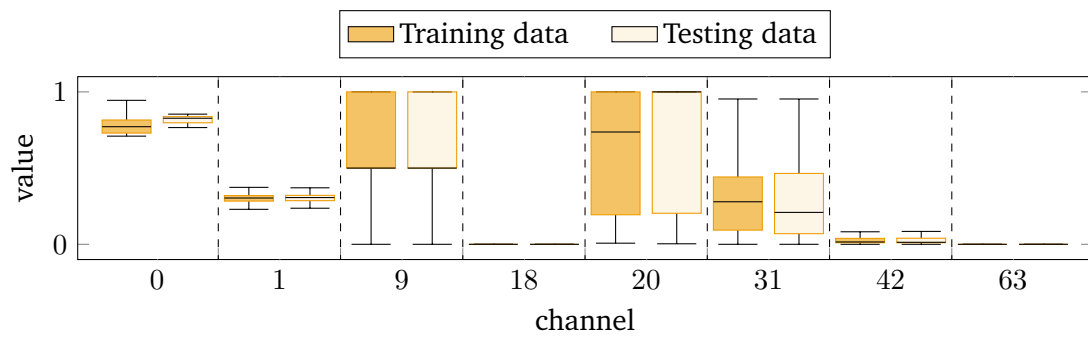


Figure 6.15: Nominal data distribution for selected channels of the WADI dataset.

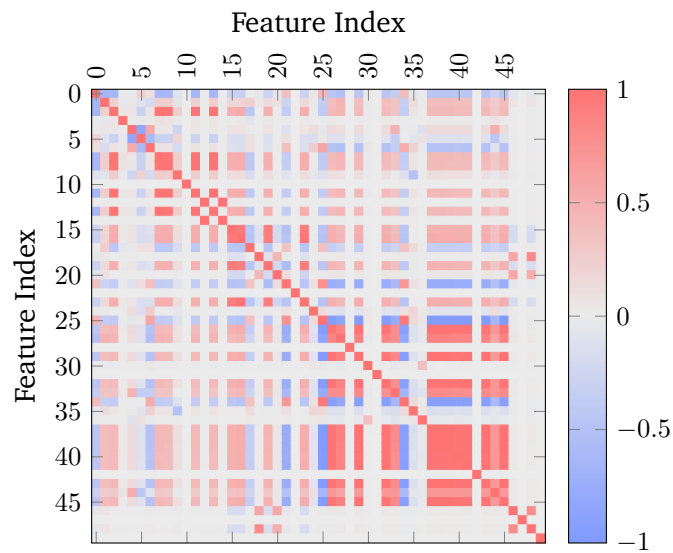


Figure 6.16: Feature correlation of the SWaT training dataset

especially when compared to the long time series in the training data. Figure 6.18a shows the length of the anomalous segments. With a maximum of 173 time steps, few and short anomalies are present, especially when compared to the other investigated datasets. The anomalies can be found at the beginning, middle, and end of the dataset (Figure 6.18b).

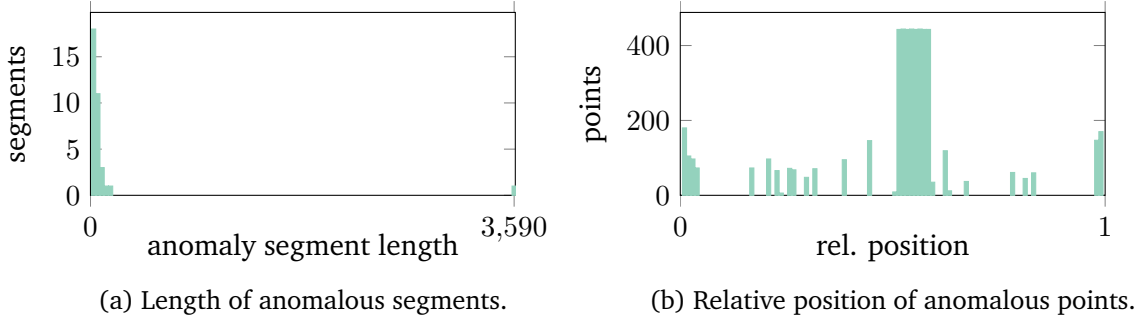


Figure 6.17: Analysis of the anomalies in the SWaT test dataset.

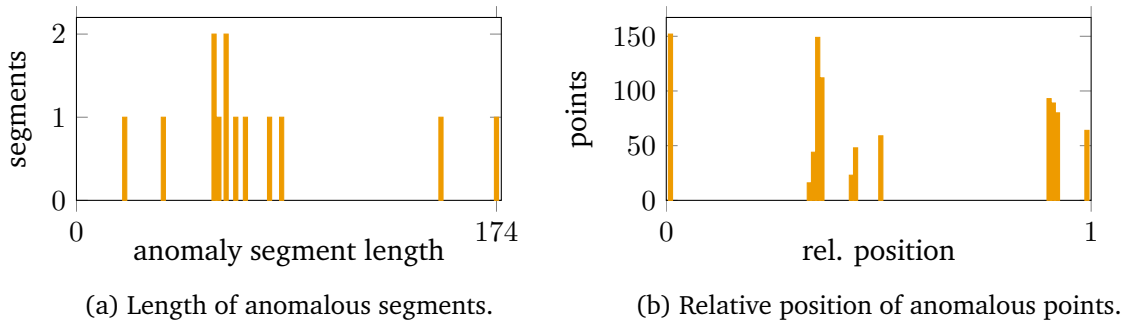


Figure 6.18: Analysis of the anomalies in the WADI test dataset.

Summary and Conclusion.

Although the presented datasets are primarily used for time series anomaly detection, some of them have significant flaws and are frequently criticized [170, 175, 176, 186].

SMAP and MSL have short entities, so long-term dependencies within the signal can hardly be determined. The relative position bias of the anomaly towards the end of the time series could be interpreted as a run-to-failure recording [186]. Only the first channel displays sensor measurements, while the other channels are encoded in binary format. While temporal correlations within the first channel are evident, spatial correlations are hardly visible. The limited length of these datasets is a criterion for excluding them.

The time series of SMD have a reasonable length for training and testing, have a feasible ratio of anomalous data points and anomaly distribution, and show temporal and spatial relations within the nominal data. For some authors, this dataset is the only one among the mentioned datasets suitable for anomaly detection [170].

The PSM dataset is characterized by its long multivariate time series with high temporal correlations across distant lags. However, the long anomalies are easy to detect, especially with the point adjustment strategy applied, resulting in a high F1-score when one data point is considered to be anomalous within the three longest anomalies (a label vector with only three entries, one within each of the longest anomalies, results in $F1_{PA} = 0.82$). With more than a quarter of the testing time steps being anomalous, it is also hard to argue that anomalies are rare events.

The SWaT and WADI datasets appear to be suitable for anomaly detection at first glance, as the testbed setup and induced anomalies yield precise sensor measurements and accurate labels. The comfortable length of the training time series of WADI, as depicted in Table 6.2, is deceptive, as the first part can be perceived as the ramp-up phase, which is not suitable for training an algorithm according to Deng et al. [187]. While most anomalies have a reasonable length and are distributed across the time series, there are uncommonly long anomalies in both datasets. Wagner et al. [170] further detected unlabeled delayed long-term effects of anomalies in both datasets, making them unsuitable for evaluating different architectures and configurations in an anomaly detection task.

The strengths and weaknesses of the presented datasets are summarized in Table 6.3. We only consider SMD for evaluating the presented models, due to the flaws in the remaining publicly available datasets.

Table 6.3: Strengths and weaknesses of publicly available datasets.

Dataset	Strengths	Weaknesses
SMAP/ MSL		• Short time series • Inaccurate labeling • One channel with sensor data • Inconsistency in data scaling
SMD	• Long time series • Spatial and temporal dependencies • Reasonable anomaly occurrence	• Delayed and long-term effects
PSM	• Long time series • Spatial and temporal dependencies	• Long anomalous segments
SWaT/ WADI	• Long time series • Reasonable anomaly occurrence • High number of channels	• Ramp-up phase in training data • Short testing dataset (WADI) • Delayed and long-term effects

The resulting F1 score of a dataset is typically reported as a mean value across all entities, which makes direct comparison difficult. In the following section, the focus is on single entities as representative examples of the dataset.

6.2 Experimental Results

The previously discussed models are used to detect anomalies in a subset of the datasets mentioned. The multi-entity datasets require separate training for each entity [170], even though recent approaches do not fulfill this requirement and treat the different entities in the multi-entity datasets as a single time series, trying to learn a general representation of the data [105, 112, 114]. Individual training sessions for all the discussed datasets and entities yield 112 separate training sessions per architecture and configuration. However, publications do not provide individual results for multi-entity datasets, but only a mean F1-score for each dataset.

To compensate for the weaknesses of the datasets, researchers have developed various techniques and strategies. A strategy is not to use all of the entities within the datasets [72, 103, 170] or omit specific datasets [115, 170] thoroughly to evaluate the architecture. Furthermore, the authors of [73, 103, 112, 187] decided to eliminate individual channels in which the anomaly does not appear in the test dataset, thereby having a more apparent distinction between nominal and anomalous time steps and reducing the problem of anomaly detection to a minimum. However, this technique requires knowledge of the test data and the anomalies, which cannot be guaranteed in real-world scenarios. Deng et al. [187] used the SWaT and WADI datasets in a downsampled version with the ramp-up phase removed, thereby diminishing their advantage of being large datasets.

The focus of this work is on specific entities to show the strengths and flaws of the presented models, methods, and techniques. We exclude datasets that seem to be unsuitable for an anomaly detection task, are poorly labeled, or are too small for the proposed models [72, 103, 170] and reduce the number of training sessions to a minimum.

The chosen datasets and entities reflect different challenges in time-series anomaly detection. They have multiple channels recording a continuous signal, and various types of anomalies are present in the test data. To be precise, we investigate the entities `machine-1-1` and `machine-1-6` from SMD. In addition to the advantages and disadvantages of the dataset listed in Table 6.3, the chosen entities show no long-term effects of anomalies in the test data and exhibit both collective and point anomalies. For `machine-1-6`, the anomalies are present in a selection of the channels, most evident in channel 0.

The performance of different models for various datasets under uniform settings and hyperparameters is presented in Section 6.2.1. The metrics presented in Chapter 5 are utilized to compare the architectures. Also, the differences in the metrics themselves are highlighted when compared side-by-side with real results. Section 6.2.2 lists various ablation studies that were performed on selected entities and show the impact of single design choices.

Figure 6.19 shows the training process of a Swin1D-AD model and the corresponding results when trained on entity machine-1-6 of SMD with an input window size of $T = 1,024$. Without further fine-tuning or addressing the specific configuration, it is evident that with ongoing training, the reconstruction loss decreases until it converges, as visualized in Figure 6.19a. A lower loss indicates a more accurate reconstruction, lower anomaly scores, and, therefore, lower thresholds, regardless of the thresholding algorithm used. The figure shows the resulting thresholds τ for the two algorithms, POT and Gap statistics, introduced earlier. Contrary to the better reconstruction results, the corresponding F1-scores in Figure 6.19b do not monotonically increase, which would indicate a better detection performance with ongoing training. Instead, the F1-scores, particularly $F1_{\text{POT}}$, decrease after about 80 epochs of training, indicating that the model overfits to the anomalies. During the initial phase of training, only coarse structures and features, such as a running mean, are reconstructed, resulting in higher anomaly scores when point anomalies are present.

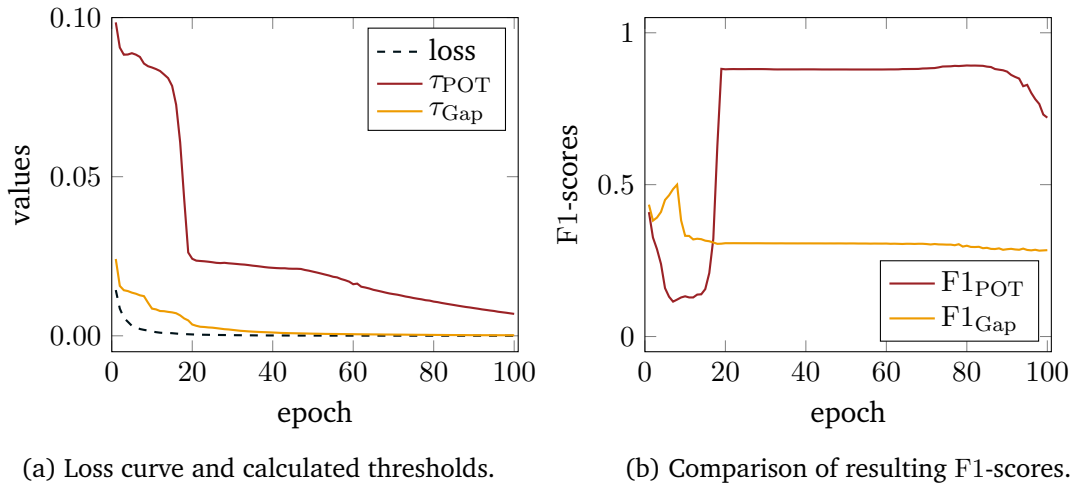


Figure 6.19: Example training progress and results of Swin1D-AD when trained on entity machine-1-6 from SMD.

6.2.1 Results

In this section, the detection scores of the different algorithms are presented under various metrics. The datasets are used as-is without any additional pre- or post-processing steps, such as normalization. The reconstruction error is calculated as the MSE between the original and the reconstructed signal at each time step.

The GAN-based model Dual-AE-GAN requires more training time to converge, so the number of training epochs is set to 1,000, while the transformer-based Swin1D-AD was trained for 20 epochs. The diffusion model EncoDiff1D is trained for 50 epochs. Since the

sampling process for the diffusion-based approach starts with random noise, the validation on the test dataset was repeated 10 times. The displayed results represent the mean values across those runs. The batch size varies across models and configurations because the input window sizes yield different numbers of training and test samples. The presented models are designed with a depth of four layers in the encoder and decoder, respectively. A critical hyperparameter is the size of the input window, which determines the context-awareness of the model and shows the robustness of the model w.r.t. the input size. The complete configuration for the different architectures is presented in Chapter 4.

Table 6.4 shows the results of the introduced models when trained on both entities of SMD separately with different evaluation strategies to calculate a threshold and the F1-score as introduced earlier. To keep the table clearly arranged, we list the mean scores across the input window sizes of $T \in \{64, 128, 256, 512, 1,024\}$ with the associated standard deviation. The complete tables and the window size dependent scores are presented in Appendix A.3.

The best and most stable results are obtained when the point adjustment technique is applied and the threshold τ_{best} is determined via an iterative search. We discussed the disadvantages of these techniques in Section 5.2. However, the results are presented here as a reference and to demonstrate the lack of comparability across different algorithms on a real example. E.g., Dual-AE-GAN and Swin1D-AD have similar $F1_{\text{PA}}$ -scores when evaluating with the POT threshold τ_{POT} . Evaluating both models on a time step level with the traditional F1-score, the GAN-based approach clearly lags (0.73 vs 0.86). The Swin1D-AD approach yields the most stable detection results across different input window sizes, as indicated by low standard deviations.

The results table reveals the effect of different thresholding techniques when the same models are used. A model can be seen as rather unsuitable with a low F1-score, while a different thresholding algorithm results in near-perfect detection scores for the same output, as can be seen within each column of the table.

Other relevant metrics for measuring model performance are AUROC and AUPRC, as presented earlier. Table 6.5 lists the corresponding values, without applying the point adjustment technique. The values underline the presented F1-scores and the efficiency of the models.

Besides the impracticality of the iterative thresholding approach in real-world scenarios, the resulting scores can serve as a reference for what is theoretically possible when the optimal threshold is found. It is optimized to find the highest F1-score without point adjustment. Impressively, the scores of the Swin1D-AD models, when using the τ_{POT} , match mostly the ones where the τ_{best} is used, revealing the strength of the model in combination with POT. The equal or similar results suggest that $\tau_{\text{POT}} \approx \tau_{\text{best}}$.

The GAN-based approaches yield better scores with *machine-1-1*, when using τ_{Gap} , with *machine-1-6*, τ_{POT} is preferable. The theoretical advantage of sparsifying the latent representation of the input in the generator is not reflected in the results of the shown entities. Training a second decoder, on the other hand, improved the detection performance of Dual-AE-GAN compared to the other approaches, even though this expansion was not used during this evaluation. An effect of the second, weaker reconstruction on the results is shown in Section 6.2.2.

The diffusion-based approach EncoDiff1D performed best when utilizing the Gap statistics to determine the threshold. The detection results improve with a larger input window size, as shown in Tables A.4 and A.6.

Due to the overly optimistic and impractical results obtained when attempting to find the best-suited algorithm for a dataset, the calculations using the point adjustment technique are excluded in the following. While the presented results use only point-level reconstruction error, model- and data-specific anomaly score calculations are presented in the next section.

Figure 6.20 visualizes a subsequence from the first channel from the test data of entity *machine-1-1* of SMD (left column) with length 1,000 and the reconstructions with different input window sizes of our presented models. While the shown part of the first channel of the time series has no labeled anomalies, it is challenging for the models to reconstruct. It exhibits a temporal pattern similar to that in the training data, but with a larger amplitude, indicating a variation in scale rather than in shape. Reconstructing this time series is difficult for the AE-GAN, Sparse-AE-GAN, and EncoDiff1D models, and they are unable to provide accurate signals. AE-GAN reconstructs the signal next to the local peak reasonably, at least with small input windows. The unseen peak seems to be cut off during reconstruction. The biggest visual improvement across input window sizes can be seen for the Dual-AE-GAN architecture. Large input windows yield greater accuracy in reconstruction. With minor variations, our Swin1D-AD model can reproduce the subsequence flawlessly across the investigated window sizes and achieves the best performance among the models, even though it took fewer epochs to train. The resulting F1-scores do not necessarily reflect the reconstruction results, e.g., for $\text{AE-GAN}_{1,024}$, where $\text{F1}_{\text{POT}} = 0.30$. Large deviations from the normal data range in anomalous segments still yield higher anomaly scores when the model only predicts the mean channel value.

Figure 6.21 shows another subsequence of the same entity with an anomalous part. While the pattern may be similar to the ones seen before, the even higher amplitude characterizes the anomaly. The models show similar behavior again. Especially our Swin1D-AD seems to be able to reconstruct anomalous parts better than the other models, which correlates with our earlier findings (cf. Table 4.6).

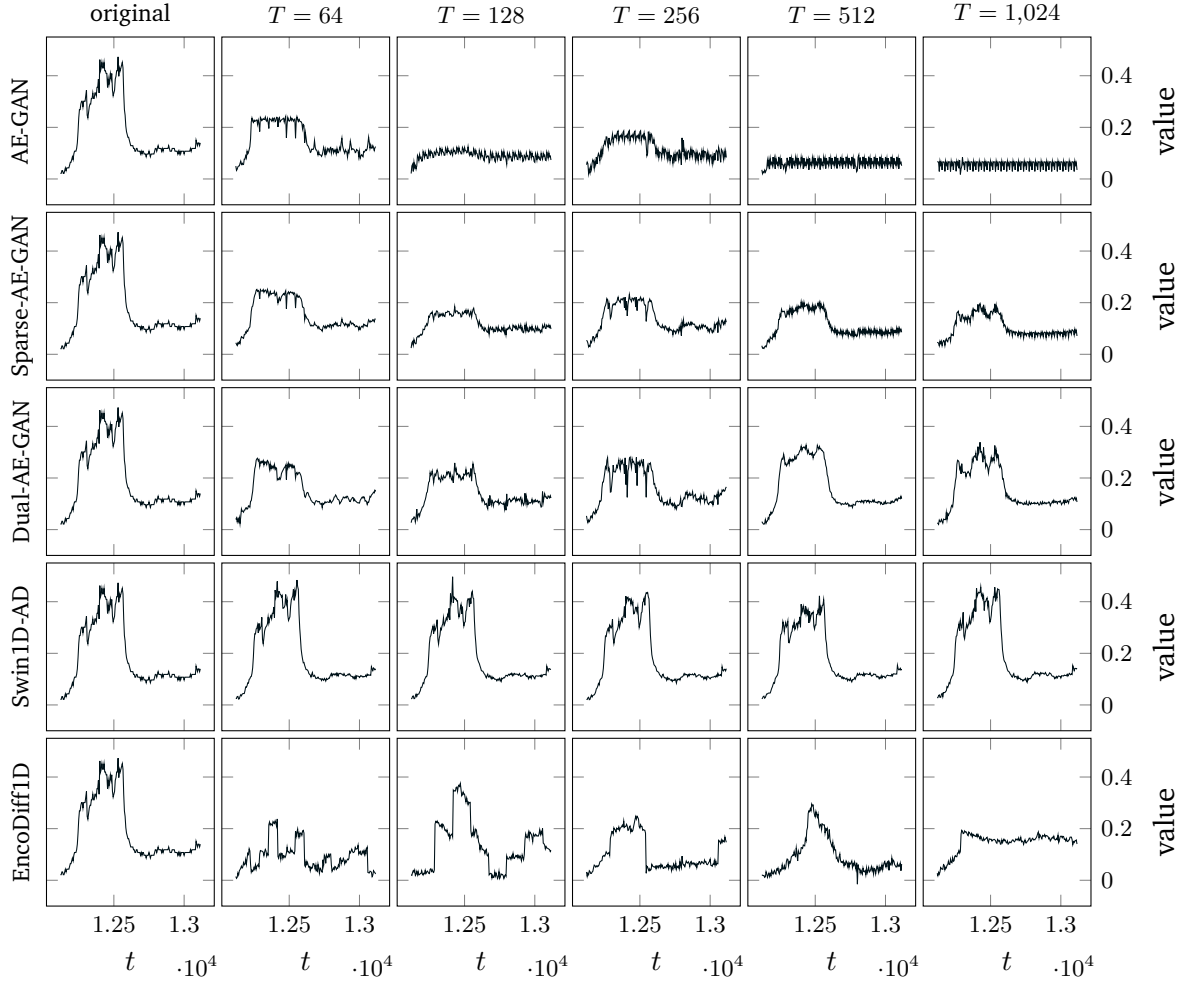


Figure 6.20: Comparison of the reconstructions on SMD machine-1-1 (without anomalies).

6.2.2 Ablation Studies

The models and results presented in the previous section serve as a baseline for the following ablation studies, which are conducted only on entity machine-1-6. The focus is on the models with an input window size of $T = 512$, evaluated with the point-wise F1-score to keep models more comparable. Furthermore, τ_{POT} is used as a threshold. While numerous adjustments and their combinations are possible, the results of the most intuitive and promising ones can be found in Table 6.6 and in the figures of this section. The table shows an **improvement** or a **worsening** of the results w.r.t. the ones reported earlier, with the best score highlighted in bold¹. First adjustments to the input data are discussed before listing modifications to the evaluation process. Different hyperparameter settings for model design and the training process, such as the selection of architectural compo-

¹Comparisons are accurate to two decimal places.

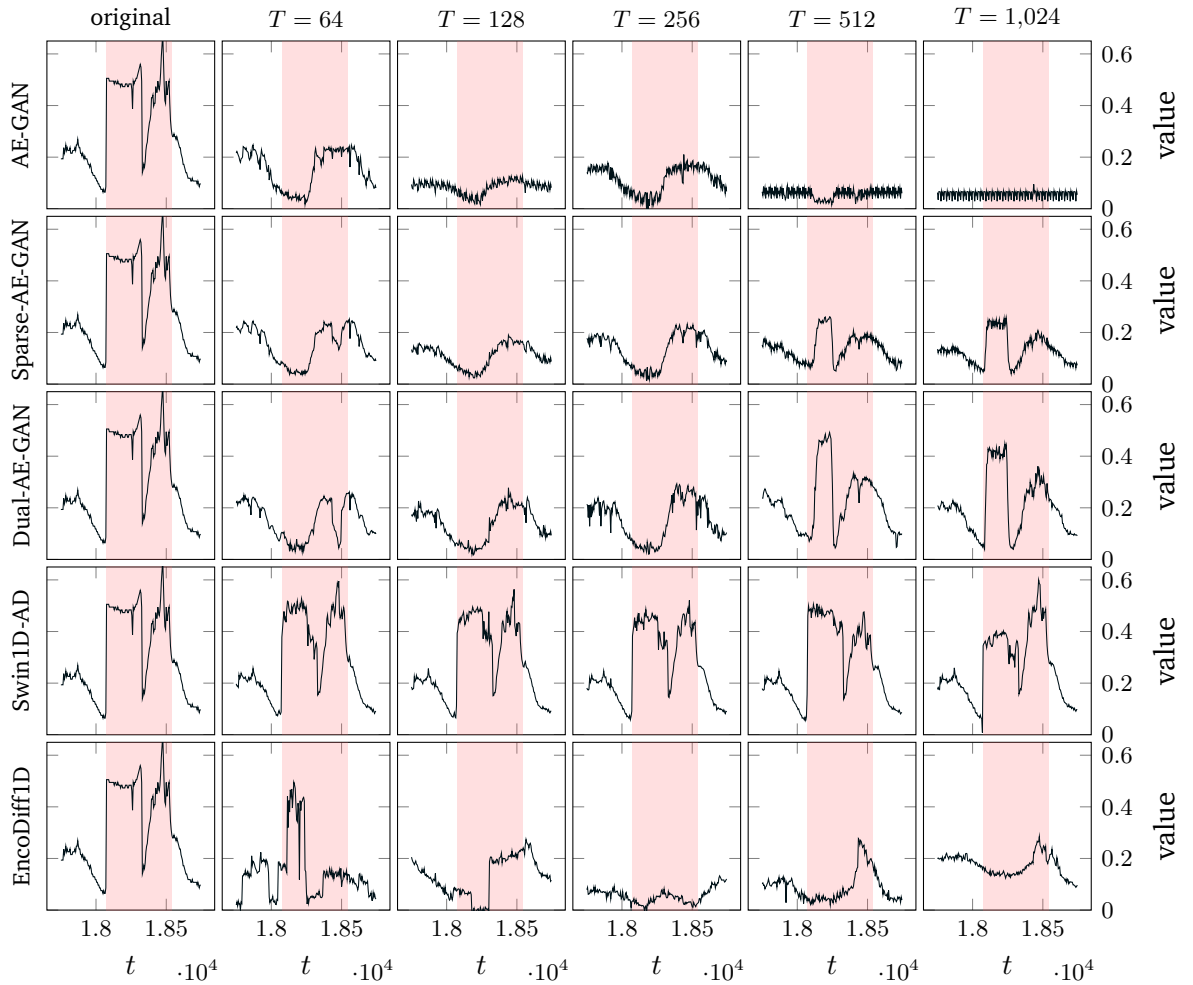


Figure 6.21: Comparison of the reconstructions on SMD machine-1-1 (with anomalies).

nents or the choice of optimizer and learning rate, are omitted and considered already optimized.

Input Data Adjustments

The used datasets and their discussed weaknesses affect the results and the interpretation of various algorithms. Several approaches have been introduced to mitigate these flaws, primarily aimed at simplifying the training objective by reducing the number of input channels of the signal. Schmidl et al. [72] analyzed numerous datasets for time series anomaly detection, including some of the previously investigated ones. They categorize the time series from SMAP and MSL as univariate, as they focus on the first channel. Other researchers have similarly reduced the number of used channels in these and other datasets to those where anomalies occur or excluded channels that only contain constant values [73, 103, 112, 114, 187].

Besides selecting the channels for model training, a common preprocessing step is to utilize z-normalization (cf. Equation (3.1)), as done in [103], to eliminate issues related to different data scales, as mentioned earlier. While this normalization technique ensures a zero-centered time series, global anomalies, such as spikes, are more easily detectable. The different time series presented are stationary, allowing for normalization of the entire time series beforehand. As indicated in the table by color, the models do not necessarily benefit from normalized data for the chosen entity. Remarkably, the detection scores of Swin1D-AD remain high across different window sizes. The F1-scores of AE-GAN and Sparse-AE-GAN are near zero. The only improvement can be seen with EncoDiff1D. Zero-centered data makes the learning and generation process of diffusion models easier and more stable, resulting in better detection results.

Evaluation Adjustments

Adjustments to the evaluation pipeline can be implemented easily, as no model retraining is needed. In addition to the various methods for determining suitable thresholds, the calculation of the anomaly score is examined in the following.

The MSE is used in the experiments as a standardized metric for point-wise reconstruction error. While it plays a crucial role in locating anomalies and therefore achieving high F1-scores, it can be combined with other model-specific metrics.

For the GAN-based approaches, only the generator output is considered during evaluation to compute the point-wise anomaly score. The discriminator’s ability to distinguish between real (normal) and fake (anomalous) samples can also be utilized as an additional input to the anomaly score, as

$$\alpha(\mathbf{X}) = \text{MSE}(\mathbf{X}, \hat{\mathbf{X}}) + \lambda_{\mathcal{D}} \mathcal{D}(\mathbf{X}) \quad (6.3)$$

with the factor $\lambda_{\mathcal{D}} \in [0, 1]$ that controls the balance between the two scores. Equally, a similarity measurement can be included in the calculation of the anomaly score. We used the adjusted cosine similarity as presented in Section 4.4.3.

In our Dual-AE-GAN approach, the generator produces two reconstructions, which can both be used to compute reconstruction errors and the anomaly score. The complete anomaly score calculations can be written as

$$\alpha(\mathbf{X}) = (1 - \lambda') \text{MSE}(\mathbf{X}, \hat{\mathbf{X}}) + \lambda' \text{MSE}(\mathbf{X}, \hat{\mathbf{X}}') + \lambda_{\mathcal{D}} \mathcal{D}(\mathbf{X}) + \lambda_{\text{sim}}(1 - \text{cos_sim}) . \quad (6.4)$$

For the configurations of AE-GAN and Sparse-AE-GAN, $\lambda' = 0$ and the mean squared error is not computed, as no second reconstruction exists. Combining the point-wise anomaly score with the similarity scores did not improve the results. When combining the MSE with the output of the discriminator in the GAN-based approaches, AE-GAN and Sparse-

AE-GAN show a slightly better F1-score, indicating that the discriminator can compare between normal and anomalous sequences. In the Dual-AE-GAN approach, the discriminator faces a more challenging task during training, which negatively impacts its ability to correctly distinguish between real and fake samples. Utilizing the second reconstruction in this setup does not affect the result.

As pointed out in Section 5.1, the anomalies are often most evident in single channels. While preserving the input dimensions, model configurations, and weights as in the original experiments, a straightforward approach is to calculate the anomaly score as the MSE for those channels only. For the given example, the first channel provides the most unambiguous indication of anomalous behavior. Most models yield the same F1-score when applying this adjustment to the evaluation pipeline.

6.2.3 Results of Baseline Approaches

In addition to our models, we list the results from baseline approaches. We use z-normalization (abbreviated as z-norm. in the table) and the PCA-Error, as described in Section 3.1, as simple and straightforward techniques. Z-normalization can be applied to the entire training dataset, since the entities are stationary. The mean and standard deviation for each channel are then applied to the test data. To calculate the PCA-Error, we reduce the number of channels from 38 to 10 and then reconstruct the signal with the inverse transform.

For more advanced deep learning approaches, we utilize AnomalyTransformer [105] (abbreviated as AnomalyTr. in the tables), TranAD [103], and ImDiffusion [114], as introduced earlier. Their published result differ from those presented here due to evaluation strategies, input window sizes, and selection of entities.

The results of these baseline approaches are listed in Table 6.7. The window-size-dependent results of the deep-learning approaches are presented in Appendix A.3.

The simple z-normalization yields the best results in our examples compared to other baseline algorithms. Especially when using POT as a thresholding algorithm, the resulting F1-score is not far behind our scores. The performance of this approach is based on the structure and types of the anomalies present in the datasets. The anomalies in machine-1-6 are often characterized by extreme values, which are highlighted by the normalization. On this entity, the z-normalization yields $F1_{POT} = 0.80$ as the best point-wise result.

The PCA reconstruction method achieves high results on the entity machine-1-6 when the optimal threshold is used $F1_{best} = 0.88$. With the preferred POT algorithm, we can achieve point-wise scores of 0.31 and 0.37 for both tested entities.

The reference algorithm AnomalyTransformer yields results comparable to, or even better than, ours when the point adjustment technique is applied, as the authors do in their

publication. When using point-wise evaluation, AnomalyTransformer is impractical, regardless of the input window size².

The performance of TranAD shows good point-wise performance when using the Gap thresholding method. The best performance, excluding $F1_{\text{best}}$, can be achieved with $F1_{\text{POT,PA}}$, which the authors use in their publication.

The diffusion-based model ImDiffusion also performs poorly in our standardized evaluation scenario. In their publication, the authors use only the test dataset to determine the optimal threshold and employ the point adjustment strategy to maximize scores.

The resulting $F1_{\text{PA}}$ -scores of all shown approaches are in line with our *near-perfect* results achieved under the same conditions and once more show that algorithms cannot be compared using this strategy.

Increasing the window size only affects the training and inference times of deep-learning-based algorithms, since standard MHA modules are used. TranAD and ImDiffusion are not trainable on our hardware setup with $T = \{512, 1,024\}$, even when the batch size is reduced.

6.3 Summary and Conclusion

Publicly available, unified datasets are the backbone for developing novel architectures, as they provide a precise quality metric for models, enabling comparison. The datasets presented in this chapter are widely used to illustrate the performance gains of new anomaly detection models. Despite the large number of datasets, only a few are suitable due to imprecise labeling, poor data quality, high anomaly rates, or small dataset sizes.

Most datasets and entities are excluded from this work. Instead, we focus on two time series from SMD that are representative of a multivariate anomaly detection problem. We trained our models on both entities with varying input window sizes, thereby showing the necessity of context-aware models when long temporal correlations exist. We compared our models against SOTA deep learning-based architectures under consistent conditions. While most publications use the polished $F1_{\text{PA}}$ -score, we use a point-wise calculation with a standardized thresholding algorithm and compare the results of different thresholding and evaluation strategies. While this setup might not yield the best anomaly scores or detection results, it highlights the strengths and weaknesses of the different models and allows them to be compared.

Our proposed models show mixed results. The GAN-based architectures AE-GAN can benefit from a sparsification of the latent space during training with Sparse-AE-GAN and from the two-decoder setup in the generator of Dual-AE-GAN. Our transformer-based

²The authors originally used $T = 100$, which yields $F1_{\text{Gap}} = 0.03$, $F1_{\text{Gap,PA}} = 0.85$ and $F1_{\text{Gap}} = 0.03$, $F1_{\text{Gap,PA}} = 0.92$ for the two entities.

architecture, Swin1D-AD, achieves the best performance in detecting anomalies across the presented entities when using the POT method to determine thresholds and calculate point-wise F1-scores. Our diffusion-based model, EncoDiff1D, performs best when evaluated using the Gap method to determine the threshold. The detection scores with POT are mostly negligible.

To put our models' performance in a clearer context, we trained and evaluated the entities using statistical and learning-based approaches. While simple approaches can outperform trained models and can therefore be a good alternative, they cannot detect contextual anomalies. Other listed SOTA approaches fail with our standardized evaluation process. A broader context window is not beneficial for those approaches, but leads to high latencies during training and inference due to their poor scalability.

In the upcoming chapter, we demonstrate the performance of Swin1D-AD in real-world scenarios.

Table 6.4: Evaluation on two entities of SMD with different methods of F1-score calculations and varying threshold algorithms. The scores represent the mean F1-scores across different input window sizes with the corresponding standard deviation. Best point-wise F1-scores for every non-iterative thresholding algorithm are highlighted in bold.

machine-1-1						
		AE-GAN	Sparse-AE-GAN	Dual-AE-GAN	Swin1D-AD	EncoDiff1D
POT	F1	0.25 $\pm$ 0.05	0.27 $\pm$ 0.07	0.16 $\pm$ 0.04	0.50 $\pm$ 0.07	0.19 $\pm$ 0.03
	F1 _{PA}	0.97 $\pm$ 0.02	0.94 $\pm$ 0.01	1.00 $\pm$ 0.00	0.85 $\pm$ 0.04	0.99 $\pm$ 0.01
	F1 _{PA%K}	0.37 $\pm$ 0.07	0.40 $\pm$ 0.10	0.26 $\pm$ 0.05	0.64 $\pm$ 0.05	0.29 $\pm$ 0.04
Gap	F1	0.42 $\pm$ 0.06	0.35 $\pm$ 0.05	0.59 $\pm$ 0.07	0.32 $\pm$ 0.02	0.35 $\pm$ 0.06
	F1 _{PA}	0.80 $\pm$ 0.05	0.81 $\pm$ 0.08	0.85 $\pm$ 0.04	0.33 $\pm$ 0.03	0.87 $\pm$ 0.04
	F1 _{PA%K}	0.57 $\pm$ 0.06	0.48 $\pm$ 0.05	0.73 $\pm$ 0.05	0.33 $\pm$ 0.03	0.50 $\pm$ 0.07
static	F1	0.17 $\pm$ 0.00	0.16 $\pm$ 0.02	0.18 $\pm$ 0.01	0.11 $\pm$ 0.00	0.16 $\pm$ 0.03
	F1 _{PA}	0.75 $\pm$ 0.02	0.75 $\pm$ 0.07	0.79 $\pm$ 0.01	0.82 $\pm$ 0.01	0.81 $\pm$ 0.06
	F1 _{PA%K}	0.26 $\pm$ 0.01	0.26 $\pm$ 0.03	0.28 $\pm$ 0.02	0.18 $\pm$ 0.01	0.24 $\pm$ 0.05
best	F1	0.60 $\pm$ 0.09	0.41 $\pm$ 0.04	0.66 $\pm$ 0.09	0.54 $\pm$ 0.06	0.46 $\pm$ 0.06
	F1 _{PA}	0.69 $\pm$ 0.08	0.59 $\pm$ 0.12	0.84 $\pm$ 0.03	0.66 $\pm$ 0.09	0.69 $\pm$ 0.10
	F1 _{PA%K}	0.66 $\pm$ 0.09	0.50 $\pm$ 0.04	0.77 $\pm$ 0.07	0.62 $\pm$ 0.07	0.57 $\pm$ 0.05
machine-1-6						
		AE-GAN	Sparse-AE-GAN	Dual-AE-GAN	Swin1D-AD	EncoDiff1D
POT	F1	0.54 $\pm$ 0.26	0.45 $\pm$ 0.15	0.73 $\pm$ 0.25	0.86 $\pm$ 0.03	0.03 $\pm$ 0.02
	F1 _{PA}	0.89 $\pm$ 0.09	0.91 $\pm$ 0.02	0.97 $\pm$ 0.03	0.98 $\pm$ 0.00	0.92 $\pm$ 0.00
	F1 _{PA%K}	0.62 $\pm$ 0.25	0.59 $\pm$ 0.18	0.80 $\pm$ 0.23	0.91 $\pm$ 0.01	0.05 $\pm$ 0.03
Gap	F1	0.44 $\pm$ 0.17	0.47 $\pm$ 0.05	0.32 $\pm$ 0.03	0.31 $\pm$ 0.00	0.68 $\pm$ 0.19
	F1 _{PA}	0.49 $\pm$ 0.26	0.52 $\pm$ 0.05	0.33 $\pm$ 0.04	0.31 $\pm$ 0.00	0.93 $\pm$ 0.04
	F1 _{PA%K}	0.47 $\pm$ 0.22	0.47 $\pm$ 0.05	0.32 $\pm$ 0.03	0.31 $\pm$ 0.00	0.76 $\pm$ 0.16
static	F1	0.25 $\pm$ 0.02	0.27 $\pm$ 0.00	0.25 $\pm$ 0.02	0.24 $\pm$ 0.01	0.19 $\pm$ 0.02
	F1 _{PA}	0.95 $\pm$ 0.01	0.94 $\pm$ 0.01	0.96 $\pm$ 0.01	0.91 $\pm$ 0.00	0.92 $\pm$ 0.02
	F1 _{PA%K}	0.36 $\pm$ 0.02	0.38 $\pm$ 0.01	0.47 $\pm$ 0.23	0.35 $\pm$ 0.02	0.29 $\pm$ 0.02
best	F1	0.89 $\pm$ 0.00	0.78 $\pm$ 0.08	0.88 $\pm$ 0.01	0.88 $\pm$ 0.00	0.78 $\pm$ 0.07
	F1 _{PA}	0.99 $\pm$ 0.00	0.90 $\pm$ 0.06	0.98 $\pm$ 0.01	0.98 $\pm$ 0.00	0.91 $\pm$ 0.05
	F1 _{PA%K}	0.93 $\pm$ 0.00	0.83 $\pm$ 0.07	0.92 $\pm$ 0.01	0.92 $\pm$ 0.00	0.85 $\pm$ 0.06

Table 6.5: Evaluation of our algorithms on two entities of SMD with varying threshold algorithms. AUROC and AUPRC scores are shown here. Best scores for every non-iterative thresholding algorithm are highlighted in bold.

machine-1-1						
		AE-GAN	Sparse-AE-GAN	Dual-AE-GAN	Swin1D-AD	EncoDiff1D
POT	AUROC	0.57 $\pm$ 0.02	0.58 $\pm$ 0.02	0.55 $\pm$ 0.01	0.71 $\pm$ 0.06	0.55 $\pm$ 0.01
	AUPRC	0.20 $\pm$ 0.01	0.19 $\pm$ 0.02	0.17 $\pm$ 0.01	0.31 $\pm$ 0.07	0.18 $\pm$ 0.01
Gap	AUROC	0.68 $\pm$ 0.03	0.63 $\pm$ 0.03	0.77 $\pm$ 0.04	0.77 $\pm$ 0.03	0.62 $\pm$ 0.03
	AUPRC	0.24 $\pm$ 0.05	0.19 $\pm$ 0.01	0.40 $\pm$ 0.08	0.19 $\pm$ 0.01	0.21 $\pm$ 0.05
static	AUROC	0.54 $\pm$ 0.00	0.54 $\pm$ 0.01	0.55 $\pm$ 0.01	0.53 $\pm$ 0.00	0.54 $\pm$ 0.01
	AUPRC	0.12 $\pm$ 0.00	0.11 $\pm$ 0.01	0.13 $\pm$ 0.01	0.11 $\pm$ 0.00	0.12 $\pm$ 0.01
best	AUROC	0.86 $\pm$ 0.04	0.74 $\pm$ 0.06	0.83 $\pm$ 0.05	0.82 $\pm$ 0.01	0.75 $\pm$ 0.06
	AUPRC	0.41 $\pm$ 0.09	0.23 $\pm$ 0.02	0.48 $\pm$ 0.11	0.35 $\pm$ 0.06	0.27 $\pm$ 0.05
machine-1-6						
		AE-GAN	Sparse-AE-GAN	Dual-AE-GAN	Swin1D-AD	EncoDiff1D
POT	AUROC	0.72 $\pm$ 0.14	0.65 $\pm$ 0.06	0.71 $\pm$ 0.14	0.87 $\pm$ 0.02	0.51 $\pm$ 0.01
	AUPRC	0.48 $\pm$ 0.20	0.40 $\pm$ 0.09	0.43 $\pm$ 0.26	0.79 $\pm$ 0.04	0.17 $\pm$ 0.01
Gap	AUROC	0.67 $\pm$ 0.10	0.76 $\pm$ 0.04	0.59 $\pm$ 0.03	0.58 $\pm$ 0.00	0.80 $\pm$ 0.11
	AUPRC	0.31 $\pm$ 0.18	0.30 $\pm$ 0.04	0.19 $\pm$ 0.01	0.18 $\pm$ 0.00	0.58 $\pm$ 0.16
static	AUROC	0.57 $\pm$ 0.01	0.57 $\pm$ 0.00	0.57 $\pm$ 0.01	0.57 $\pm$ 0.00	0.55 $\pm$ 0.01
	AUPRC	0.23 $\pm$ 0.01	0.23 $\pm$ 0.01	0.24 $\pm$ 0.01	0.21 $\pm$ 0.01	0.19 $\pm$ 0.01
best	AUROC	0.90 $\pm$ 0.00	0.86 $\pm$ 0.04	0.91 $\pm$ 0.00	0.90 $\pm$ 0.00	0.87 $\pm$ 0.03
	AUPRC	0.83 $\pm$ 0.01	0.65 $\pm$ 0.12	0.82 $\pm$ 0.03	0.82 $\pm$ 0.00	0.66 $\pm$ 0.10

Table 6.6: Resulting F1-scores of different ablation studies discussed. **Green**, black, and **red** indicate performance relative to the standard models, listed as *baseline* here. Experiments were conducted on SMD machine-1-6. The best score of each adjustment is highlighted in bold.

	AE-GAN	Sparse-AE-GAN	Dual-AE-GAN	Swin1D-AD	EncoDiff1D
baseline	0.88	0.60	0.89	0.88	0.05
first channel only	0.89	0.88	0.89	0.88	0.88
z-normalization	0.06	0.01	0.70	0.77	0.20
similarity score	0.73	0.19	0.84	0.75	0.00
discriminator score	0.89	0.62	0.44	-	-
second reconstruction	-	-	0.89	-	-

Table 6.7: Evaluation on two entities of SMD with different baseline methods of F1-score calculations and varying threshold algorithms. The scores represent the mean F1-scores across different input window sizes with the corresponding standard deviation. Best point-wise F1-scores for every non-iterative thresholding algorithm are highlighted in bold.

		machine-1-1				
		z-norm.	PCA-Error	AnomalyTr.	TranAD	ImDiffusion
POT	F1	0.38	0.31	0.01 $\pm$ 0.01	0.03 $\pm$ 0.00	0.01 $\pm$ 0.00
	F1 _{PA}	0.78	0.43	0.97 $\pm$ 0.03	0.84 $\pm$ 0.00	0.73 $\pm$ 0.00
	F1 _{PA%K}	0.53	0.36	0.02 $\pm$ 0.01	0.05 $\pm$ 0.00	0.01 $\pm$ 0.00
Gap	F1	0.32	0.25	0.02 $\pm$ 0.01	0.58 $\pm$ 0.02	0.06 $\pm$ 0.01
	F1 _{PA}	0.47	0.25	0.89 $\pm$ 0.05	0.62 $\pm$ 0.04	1.00 $\pm$ 0.00
	F1 _{PA%K}	0.41	0.25	0.03 $\pm$ 0.02	0.62 $\pm$ 0.03	0.10 $\pm$ 0.02
static	F1	0.17	0.17	0.17 $\pm$ 0.00	0.17 $\pm$ 0.00	0.17 $\pm$ 0.00
	F1 _{PA}	0.17	0.17	0.17 $\pm$ 0.00	0.17 $\pm$ 0.00	0.18 $\pm$ 0.00
	F1 _{PA%K}	0.17	0.17	0.17 $\pm$ 0.00	0.17 $\pm$ 0.00	0.18 $\pm$ 0.00
best	F1	0.08	0.30	0.01 $\pm$ 0.01	0.04 $\pm$ 0.00	0.26 $\pm$ 0.04
	F1 _{PA}	0.78	0.97	0.85 $\pm$ 0.26	1.00 $\pm$ 0.00	0.60 $\pm$ 0.23
	F1 _{PA%K}	0.13	0.43	0.01 $\pm$ 0.01	0.07 $\pm$ 0.00	0.36 $\pm$ 0.07
		machine-1-6				
		z-norm.	PCA-Error	AnomalyTr.	TranAD	ImDiffusion
POT	F1	0.80	0.37	0.01 $\pm$ 0.01	0.22 $\pm$ 0.03	0.01 $\pm$ 0.00
	F1 _{PA}	0.94	0.39	0.75 $\pm$ 0.38	0.92 $\pm$ 0.00	0.22 $\pm$ 0.00
	F1 _{PA%K}	0.87	0.39	0.02 $\pm$ 0.02	0.32 $\pm$ 0.04	0.02 $\pm$ 0.00
Gap	F1	0.31	0.30	0.03 $\pm$ 0.01	0.31 $\pm$ 0.00	0.04 $\pm$ 0.00
	F1 _{PA}	0.31	0.30	0.94 $\pm$ 0.03	0.31 $\pm$ 0.00	0.99 $\pm$ 0.00
	F1 _{PA%K}	0.31	0.30	0.04 $\pm$ 0.01	0.31 $\pm$ 0.00	0.06 $\pm$ 0.00
static	F1	0.27	0.27	0.27 $\pm$ 0.00	0.27 $\pm$ 0.00	0.27 $\pm$ 0.00
	F1 _{PA}	0.27	0.27	0.27 $\pm$ 0.00	0.27 $\pm$ 0.00	0.28 $\pm$ 0.00
	F1 _{PA%K}	0.27	0.27	0.27 $\pm$ 0.00	0.27 $\pm$ 0.00	0.28 $\pm$ 0.00
best	F1	0.78	0.88	0.02 $\pm$ 0.01	0.87 $\pm$ 0.01	0.57 $\pm$ 0.08
	F1 _{PA}	0.95	0.98	0.98 $\pm$ 0.01	1.00 $\pm$ 0.00	0.77 $\pm$ 0.12
	F1 _{PA%K}	0.87	0.92	0.03 $\pm$ 0.02	0.93 $\pm$ 0.00	0.68 $\pm$ 0.10



CASE STUDY: REAL-WORLD INDUSTRIAL DATA

Although publicly available datasets are considered real-world, they are typically obtained from a testbed or by monitoring devices for a limited period. The second case study, presented in this chapter, focuses on data from real industrial machines that were monitored for an extended period, potentially revealing more information about the degradation of components, which becomes noticeable through long-term trend shifts. Industrial machines are manufactured to run for years or decades, and a slow degradation can only be visible when different parts are monitored simultaneously over a long period.

The data from two different machines of industrial partners form the subsequent datasets discussed in this section. Section 7.1 describes the basic structure of the components to be inspected and the sensor setup. Section 7.2 provides an introduction to the datasets derived from monitoring these. The introduced networks were trained on the datasets with different settings. As no *real* anomalies occurred during the monitoring, we injected synthetical anomalies for to test our models. The anomalies are described in Section 7.3.

The detection capabilities of the models are provided in Section 7.4 before we conclude this chapter in Section 7.5.

Note: Due to the sensitivity of industrial data, raw time series data is not displayed here. However, adjusted data (normalized, noise injected, smoothed, and without axis labeling and channel mentioning) can be presented.

7.1 Setup and Objective

For anomaly detection, it is necessary to monitor components or machines with appropriate sensors. In both real-world scenarios, the machine operates continuously around the clock, with minimal pauses or downtime. This section provides a brief description of the two objectives and setups. Both use cases are part of ongoing research projects, some of which are still in progress. Most of the sensors were installed and put into operation during the respective project.

KIWA

With the first industrial partner, *Mühlbauer AG*, the objective is to monitor a *Direct Chip Attach DDA 40000-P* (DDA40K) system, which was developed as a production solution for paper antennas. In particular, our primary focus is on a single process step and component within the machine. In the pre-bonding process, bare die chips are applied to a moving substrate tape. The *die ejector* uses a needle to eject individual chips from a wafer, as visualized in Figure 7.1¹. This punch-out process can be performed up to 40,000 times per hour (approx. 11 Hz).

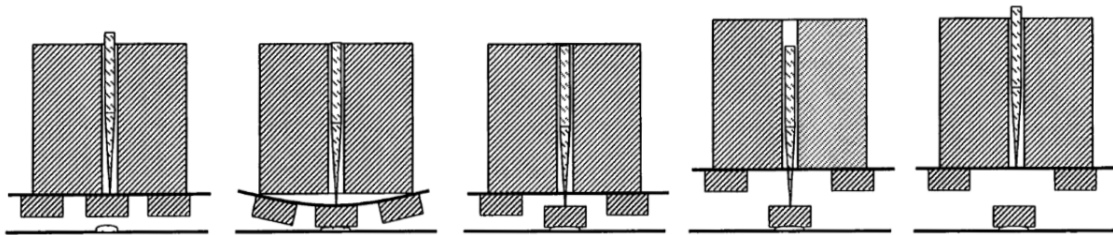


Figure 7.1: The process of ejecting a die from a wafer to a substrate.

The machine is equipped with a sensor that can measure accelerations along three axes at up to 10 kHz (hereafter referred to as *acc1*). While the first sensor is attached near the die ejector, a second platform measures accelerations at up to 24 kHz at its suspension (later referred to as *acc2*). Furthermore, machine control variables are provided via an

¹The copyright of the figure remains with Mühlbauer AG, as it is derived from the corresponding patent [188].

industrial communication standard, allowing the monitoring of the current status. The variables include the positions of different parts of the die ejector (cf. [189, 190]). A list with all the variables can be found in Appendix B.1. The DDA40K is equipped with an internal terminal computer that provides data for acc1 and machine variables.

This dataset is referred to as *KIWA*, named after the project from which this data was acquired. With this hardware setup, various problems were encountered that could not be solved completely:

1. **Data synchronization:** Due to the amount of high-frequency data, different sampling rates, and different sensor systems, efficient data synchronization is challenging. While time synchronization protocols such as Network Time Protocol (NTP) and Precision Time Protocol (PTP) exist, ensuring synchronization with millisecond and nanosecond accuracy, respectively, remains nontrivial, as other nonlinear delays in the data recordings are present. The sensor acc1 is permanently integrated into the machine and is triggered by an internal software setup, which causes delays as it loops through the system.
2. **Data loss:** For all the variables of the machine data, a value is reported when it changes, resulting in a significantly lower amount of data and thus time and memory savings for reading and writing the data to a database. The missing values can be easily filled by propagating the last valid observation to the next one. However, we observed data losses in these channels, where value changes are not recorded, thereby causing problems in the latter.

Nevertheless, models should be able to detect anomalies during die ejection to identify failures at an early stage, even when not all features are utilized.

DigiSeal

The second partner is a seal manufacturer. Two mechanical seals were installed in an industrial agitator, one at the drive end (DE) and one at the non-drive end (NDE). Each seal is equipped with several sensors that measure accelerations (A) and temperatures (T). Those sensors measure at a frequency of 1 kHz and are further denoted as *sensor_setup1*. Additionally, sensors are mounted to monitor the barrier fluid before and after it passes the seal. In detail, accelerations (A), temperatures (T), and pressures (P) are measured by *SmartSeal* sensor systems once every eight seconds [191]. Those sensors are referred to as *sensor_setup2*. The different channels are detailed in Appendix B.2 and are used anonymously throughout the main body of this work, so the exact position and measurement of the sensor remain undisclosed. To determine if the agitator is in operation, a plate with characteristic carvings was mounted on the rotation axis. This plate enables the measurement of axial displacements (Ax) and, due to the carvings, the rotating frequency (spd) of the agitator is calculated. A schematic drawing of the setup is shown in Figure 7.2.

The external sensors enable a data collection pipeline that is independent of the agitator's current status and process. As the seals are newly manufactured, there should be no degradation or significant leaks in the early stage of recording, yielding data that can be used to train our anomaly detection models. This dataset is referred to as *DigiSeal* in the following sections.

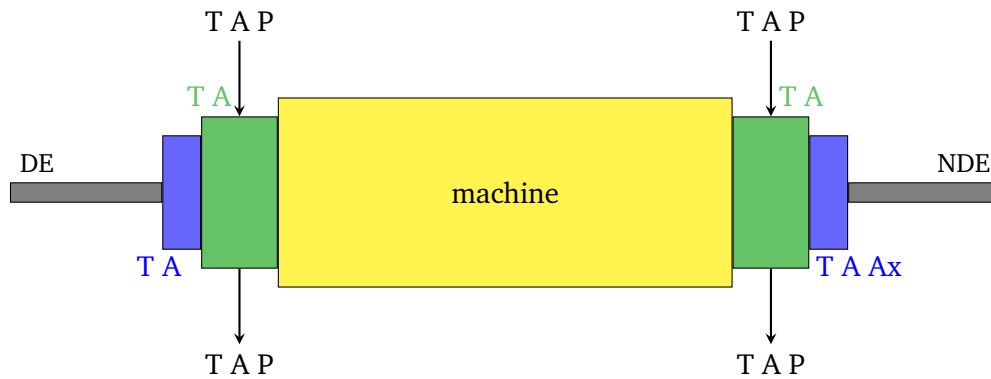


Figure 7.2: Schematic sensor setup of DigiSeal. Different colors indicate the different components: **machine**, **seal**, **bearing**, and **suspension with rotation axis**. The flow of the fluid is visualized with arrows.

As the agitator is a closed system mounted in a scaffolding structure adjacent to other machines, it must be assumed that the measured accelerations not only include the agitator with the observed seals but also those of the adjacent appliances. To prepare for seal replacement at an early stage, a leakage in the seals should be detected. According to domain experts, the most critical channels are the barrier fluid's temperature and pressure.

While the KIWA dataset monitors a single process, the DigiSeal dataset tracks the agitator, which undergoes several different processes over the course of a week. However, these specific processes and their transitions are not directly observed or recorded.

7.2 Data Inspection

As with publicly available datasets, the first step in developing an anomaly detection system is to inspect the data in order to identify features and determine helpful preprocessing steps. In contrast to previous datasets, real-world observations often lack labeled data for testing.

KIWA

For the KIWA dataset, the primary focus is on data from acc2, as it performs most reliably across the sensors used. With the current setting, the sensor has a sampling rate of 1.7

kHz, resulting in 100,000 data points per axis and minute. The data is provided via a project-internal Application Programming Interface (API) that loads data from a time series database². For more performant loading of data during training and for offline training, the data from acc2 is stored locally in the Feather format³. The dataset spans 257 days (May 2022 – March 2023). It is segmented into one-minute intervals, saved separately to enable memory-efficient, file-wise loading, efficient shuffling during batch creation, and local offline training. With this data, training, validation, and testing datasets can be predefined. As the most critical parts of the machine were changed before recording, no degradation or anomalous patterns due to malfunctions should be visible. The training dataset should consist of data representing the nominal state only, which is more likely to be present in the early days of recording. The training and validation dataset consists of snippets with a length 2,000 time steps recorded from May to August. The training data consists of 7,640 of those subsequences that yield training samples of length T .

Analyzing the data with techniques similar to those used on public datasets uncovers valuable insights. First, the autocorrelation function according to Equation 6.2 is computed for the three axes using the raw data. Figure 7.3 visualizes clear temporal correlations across the axes. A peak-detection algorithm reveals the visually apparent high correlations every 842 time steps. This value is essential when mapping a complete event into a single sample for the deep learning pipeline, as the input window length must be defined accordingly. Peak detection was further applied to the raw data, with a minimum peak distance equal to the window length, yielding starting points for sampling. Since most samples exhibit similar patterns, trained models should be able to generate a robust reconstruction.

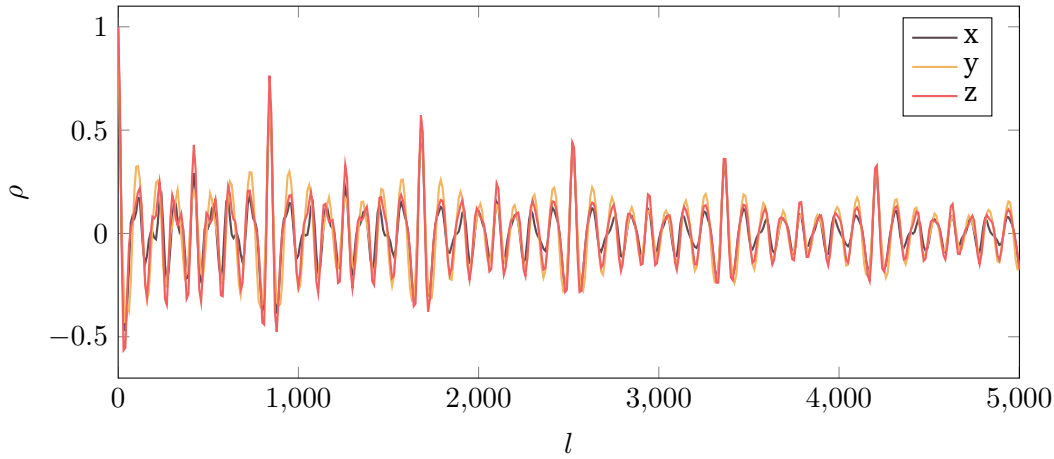


Figure 7.3: Autocorrelation functions of each channel of the three-axis acceleration sensor.

²In this project, a *Clickhouse* database was used.

³<https://github.com/wesm/feather>

Figure 7.4 illustrates the zero-centered data of the three axes during punch-out processes. The data is smoothed by using a convolutional operator with a kernel size of 20 as a rolling mean for better visualization. The overlay of ten time series reveals the correlations and similarities across the punch-out processes. One characteristic time series for every axis is thereby visualized in color. Notably, there is still some jitter after approximately 200 time steps, right before the punch-out event. As the sensor is mounted on the die ejector suspension, additional pulses are visible.

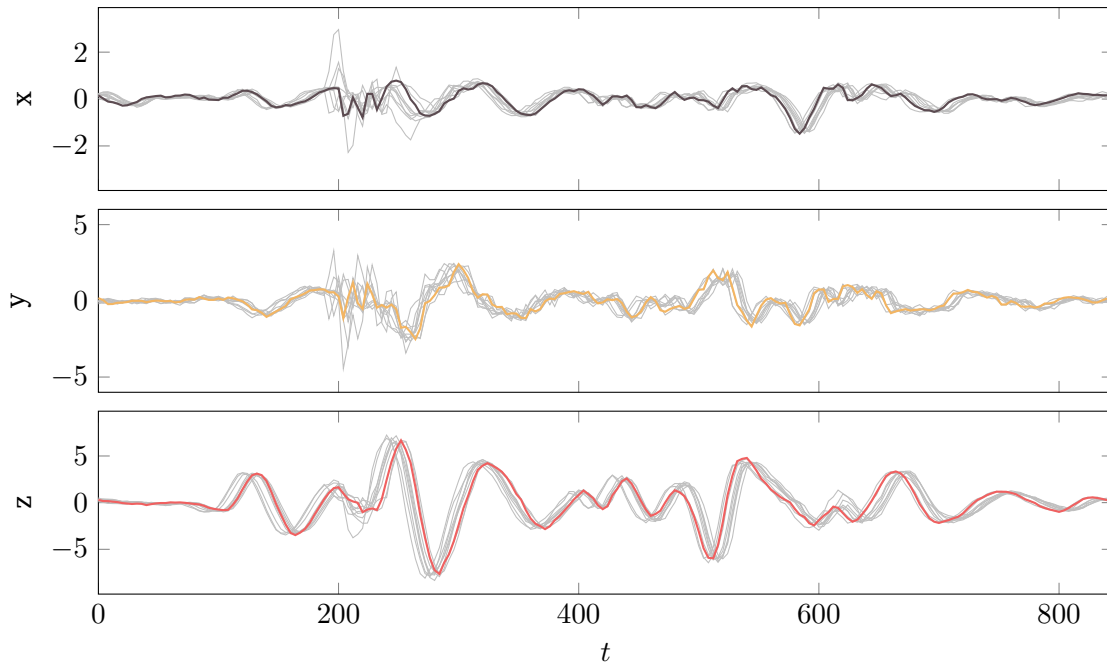


Figure 7.4: Overlay of multiple punch-out processes in the KIWA dataset.

For the illustrations, a single file of the predefined dataset was used. However, similar behavior is observed when investigating the majority of the saved data. The peak detection algorithm found the first significant peak in the autocorrelation function after 820 – 1,100 lags on a daily mean, indicating slight speed variations in the observed machine.

Due to the ability of our novel architectures to compute large contexts, a complete punch-out process is mapped into a single training sample. The resulting high correlation between samples enables the model to quickly adjust its weights.

DigiSeal

In the second use case, all the sensors were attached without interfering with the machine itself. Therefore, all the data can be stored without significant temporal delays and the need for additional machine-specific software setup in an external data collection system. Access is provided via Microsoft Azure Blob Storage, where the data can be gathered in

Parquet file format⁴. Each file in `sensor_setup1` contains ten minutes of continuous data, resulting in 600,000 data points per channel. To handle the raw data and reduce noise in the acceleration sensors, the signal is downsampled to 1 Hz using a windowed root-mean-square. Using `sensor_setup2`, the system records one measurement every eight seconds. The files from the blob storage contain continuous data of variable length and can be displayed as 10,800 points per channel and day. The (downsampled) data is also stored locally for a well-performing training pipeline, mirroring the earlier data handling setup.

The correlations between each channel are visualized in Figure 7.5. Here, data from a particular day in June 2025 was used, when the machine was running. Some clusters, i.e., darker red areas, are visible, indicating high correlation. When focusing on `sensor_setup1`, the correlations are observed between the different temperature measurements and within accelerations. The sensor measuring feature 12 is not functioning correctly and therefore does not show correlations with other features. Within `sensor_setup2`, the correlations within the same sensor type are most striking. Inter-sensor correlations are also visible, as both systems measure the same or similar parameters at different locations. For example, vibrations are transmitted throughout the entire system. Besides feature 12, the sensor for feature 17 has an internal loose connection, resulting in irregular, rare sampling. Those two channels are omitted during training and validation. The DigiSeal dataset, therefore, consists of $c = 34$ channels.

As the temperature and pressure of the barrier fluid are measured before (*Inlet*) and after (*Outlet*) it passes through the seal, it reveals direct information about the current status of the seal. Figure 7.6 displays the absolute differences between the data from the temperature sensors of `sensor_setup2`⁵. The visualized deltas are calculated by comparing the inlet and outlet temperatures at both DE and NDE, as well as by comparing the inlet and outlet temperatures with each other. The temperature differences are generally low and relatively constant, especially when computing the delta between the two inlet temperatures (ΔT_{In}), as the barrier fluid comes from the same source. The differences in outlet temperatures are irregular as the distribution of heat and substances inside the agitator changes. In the bottom plot of Figure 7.6, the delta between the inlet and the outlet at the DE is on a higher level.

We omit the autocorrelation graphs for each channel here. The temperature and pressure of the barrier fluid do not change rapidly. The axial displacement and, therefore, the calculated rotation speed are process-dependent parameters. The process order is not fixed, and the data is not labeled accordingly in order to separate them clearly.

We use the combined data from both sensor setups as input features for our model. Unlike publicly available datasets, the DigiSeal dataset consists of downsampled raw sensor

⁴<https://parquet.apache.org/>

⁵The daily temperatures were normalized to a range of $[0, 1]$ before computing the deltas for the shown figure.

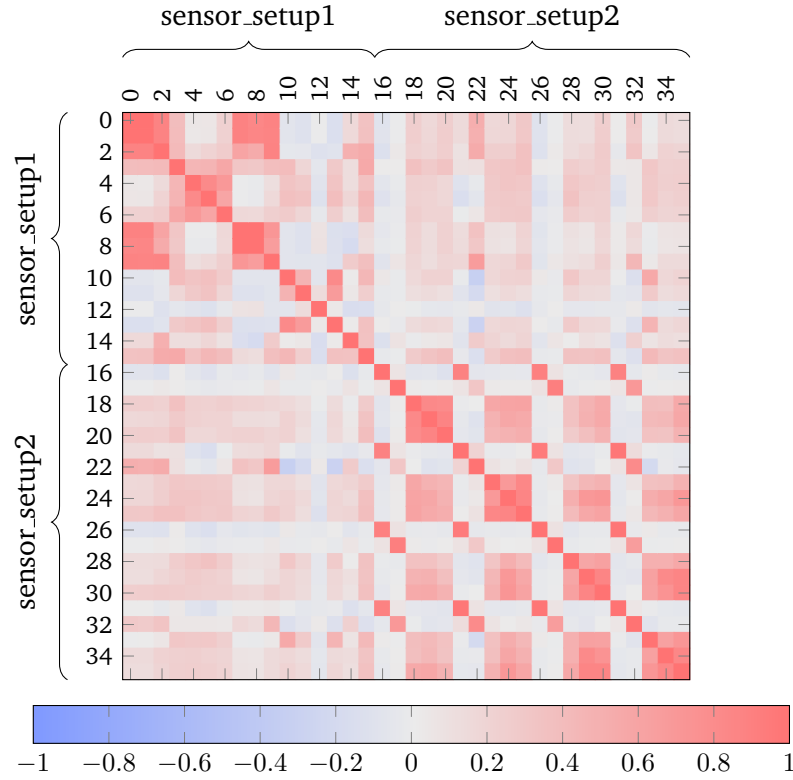


Figure 7.5: Feature correlation of the DigiSeal dataset (one day).

values, resulting in different value ranges for each channel. Before training, we compute global statistics of the training data and use them to normalize each channel individually to $[-1, 1]$ for more stable training. We chose to use the entire data of June 2025 for training and validation, as we observed stable behavior in our monitoring setup during that month.

7.3 Anomaly Injection

As we do not have actual anomalous data, we need to inject synthetic anomalies to evaluate the model's performance. Different types of anomalies were already introduced in Section 2.2.1.

We simulate the following anomalies for our samples:

- **Point anomaly:** We simulate a point anomaly by setting $\mathbf{X}_t \leftarrow \mu(\mathbf{X}) + 4\sigma(\mathbf{X})$ at time step t .
- **Contextual shift:** We shift the values of a subsequence of the input sample by a given value.

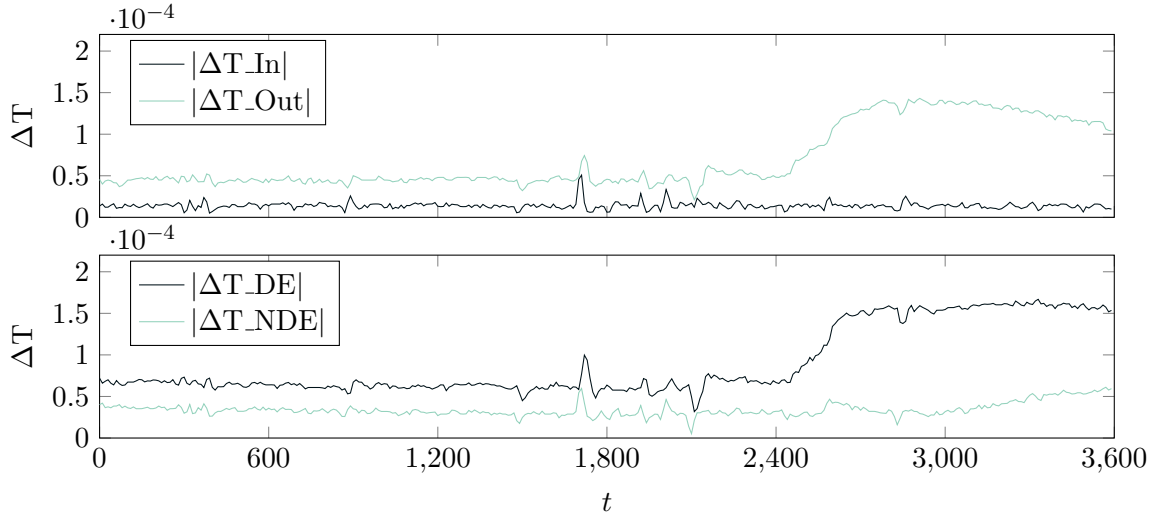


Figure 7.6: Absolute differences of monitored temperatures of sensor_setup2. Top: Differences of inlet and outlet temperatures. Bottom: Differences at DE and NDE.

- Contextual replacement: Replacing a subsequence with another normal-looking one requires the model to think in a broader context. We copy a portion of the signal and place it at a random position within the sample.
- Drift: Sensor values might drift in real-world scenarios, which can be seen as a trend anomaly for a limited interval.

We keep the injection position random in our examples and present the time series in the following section.

7.4 Experimental Results

Both datasets are stored locally for efficient training. Due to missing labels in the raw data, the standardized metrics for measuring model performance cannot be used. However, we can assess the model's reconstruction capabilities and use them as a quality metric. Known machine failures and synthetic anomalies complement the experiments.

We chose Swin1D-AD for the reconstruction task on both datasets and trained the model for one and ten epochs, respectively. Due to the large amount of data and the repeating patterns in the KIWA dataset, a single epoch already yields accurate reconstruction results. As demonstrated with the publicly available datasets, the model is characterized by

1. its scalability with respect to input window sizes,
2. stable training,
3. a high reconstruction accuracy with limited training time,
4. fast inference times that are suitable for real-world applications.

The validation data is used as a reference for the model’s capacity to reconstruct data. We trained our model with a range of input window sizes, as also used before, to highlight the benefits of large windows in real-world scenarios. Table 7.1 shows the mean reconstruction errors (computed as the MSE) across all used channels for both datasets when only normal data points are present. Although the results from the KIWA and DigiSeal datasets are not directly comparable, a decrease in reconstruction error with increasing window size is evident. Our model performs best on the KIWA dataset with an input window size of $T = 512$, which almost correlates with the length of a punch-out process. The DigiSeal dataset does not require large inputs, but our model still benefits slightly from richer context and performs best with $T = 512$.

Table 7.1: Mean reconstruction error during validation of real-world datasets (normal data only).

	$T = 64$	$T = 128$	$T = 256$	$T = 512$	$T = 1,024$
KIWA	$3.1 \cdot 10^{-4}$	$5.2 \cdot 10^{-5}$	$1.6 \cdot 10^{-4}$	$1.4 \cdot 10^{-5}$	$3.1 \cdot 10^{-4}$
DigiSeal	$1.7 \cdot 10^{-3}$	$1.9 \cdot 10^{-3}$	$1.9 \cdot 10^{-3}$	$1.4 \cdot 10^{-3}$	$1.6 \cdot 10^{-3}$

We omit visualizations of the reconstructions on the KIWA dataset because the outputs of all our models are near-perfect. We inject several synthetic anomalies into a test sample and evaluate the model’s performance in the following. Figure 7.7 shows the original sample and our altered versions, denoted as $\mathbf{X}$, as well as the reconstruction attempts by Swin1D-AD₅₁₂ as $\hat{\mathbf{X}}$. The highlighted anomalous part cannot be reconstructed accurately. We insert the anomalies presented in the previous section randomly and obtain a mean reconstruction error of 0.012 for anomalous time steps. Approximately 4% of the time steps were infected, corresponding to the anomaly ratio found in SMD.

While the overall reconstruction performance of the DigiSeal dataset is worse than for KIWA, we still see a tradeoff when anomalies are injected. Figure 7.8 visualizes an infected sample. We can obtain a mean reconstruction error of anomalous time steps of 0.004.

While training models is always performed with GPU support, inference is mainly performed on Central Processing Unit (CPU), as resources on embedded devices are limited and using a cloud solution is costly and requires the disclosure of sensitive data to third parties. The models for the two mentioned datasets run directly on the machine or on a small computer that comes with the machine to the customer. Conducted tests were made on consumer hardware as described in Section 4.1. Speed measurements were performed on both CPU and GPU and are listed in Table 7.2. The inference times refer to the number of samples of length $T = 512$ processed in one second.

Our model can reconstruct data in real time when given high-frequency data, e.g., accelerations in the KIWA dataset. The Swin1D-AD model can process approximately 32,250 time steps per second on a consumer hardware setup without utilizing GPU support. The

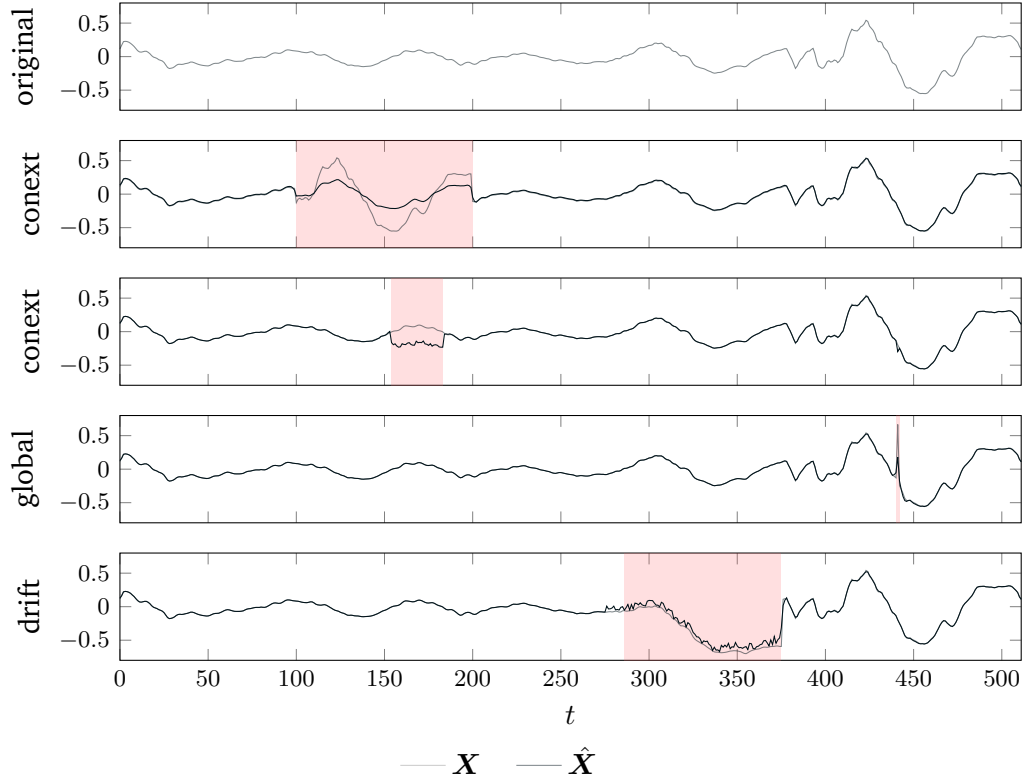


Figure 7.7: KIWA: Synthetical anomalies and their reconstructions by Swin1D-AD₅₁₂. Anomalies are in the highlighted areas.

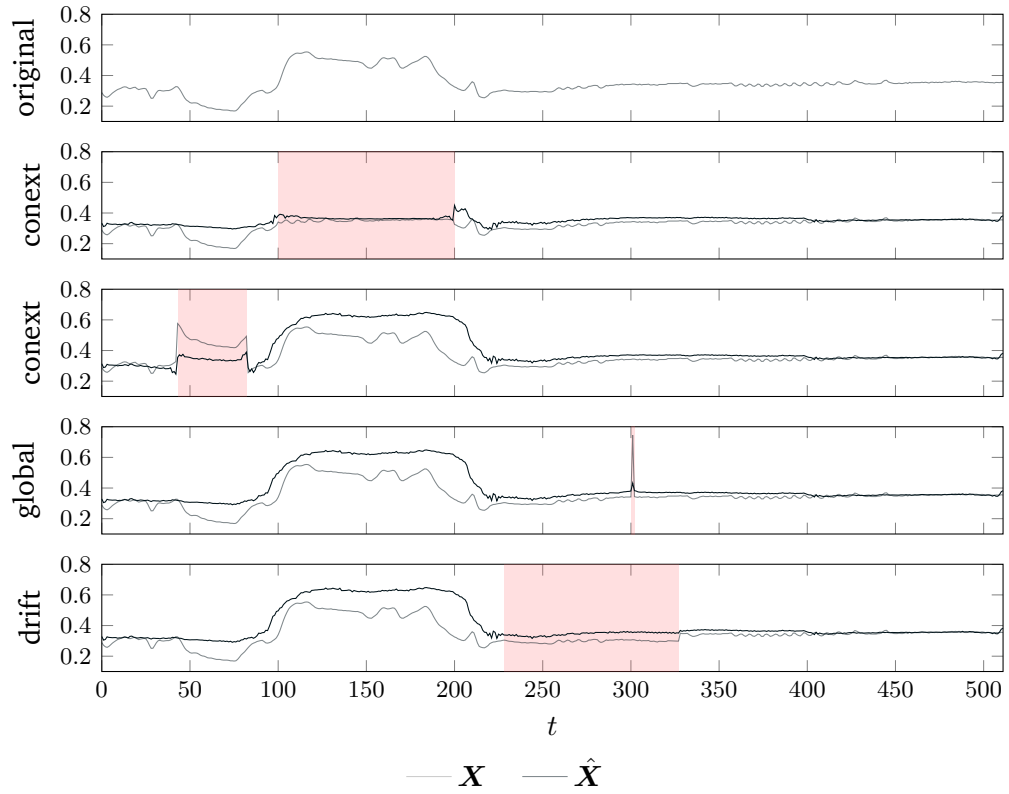


Figure 7.8: DigiSeal: Synthetical anomalies and their reconstructions by Swin1D-AD₅₁₂. Anomalies are in the highlighted areas.

DigiSeal dataset does not require high sampling frequencies. However, the table shows that the model’s performance does not drop when more input channels are used.

Table 7.2: Mean inference times of Swin1D-AD₅₁₂ in samples per second on real-world datasets.

	GPU	CPU
KIWA	105.8	88.5
DigiSeal	96.0	83.0

7.5 Summary and Conclusion

We investigated two datasets recorded in real-world scenarios with varying setups and challenges.

With the KIWA dataset, we monitored a repeating punch-out process, focusing on sensor data recorded at high sampling rates, particularly from acceleration sensors. To split the data into manageable training, validation, and test sets, we selected subsequences of the monitored time frame and used the early data to train the model. The early data comes from components in the machine that are likely to have no defects or degradation, and therefore resemble the machine’s nominal state. To map a punch-out process into a single input window, we need a model that is capable of doing so. Our presented Swin1D-AD has several advantages beyond its scalability in input dimensions. Its ability to learn data representations quickly, thereby yielding accurate reconstructions of the nominal input, lies at the foundation of successfully building and deploying a proof-of-concept version for an anomaly detection setup. With its fast inference times, it is furthermore suitable in high-frequency environments.

With the DigiSeal dataset, we monitored machine seals during production with various sensors to detect leaks. We collected data with two independent sensor setups that record machine state (vibrations, agitator speed, temperatures) and the condition of the seal and barrier fluids (temperatures, pressures). The main challenge, unlike KIWA, is that we monitor several consecutive processes that are unlabeled, have irregular durations, and are not clearly separable. Due to the sensors used, their mounting positions, and the closed system, we observe strong correlations within the same sensor types and between physically connected values.

We also utilize Swin1D-AD with the DigiSeal dataset. Our experiments revealed that long context windows are beneficial for reconstruction performance, even though the input data is downsampled to eight seconds, yielding more than two hours of data with our largest input window.

Overall, our Swin1D-AD model with an input window size of $T = 512$ showed the best reconstruction abilities across the tested setups. We injected synthetic anomalies into the test data to visually showcase the model’s performance in reconstructing and, therefore, detecting them. Our model shows clearly better results on the KIWA dataset and performs well on the DigiSeal data.

8

SUMMARY, CONCLUSION AND OUTLOOK

This work examines the potential of deep learning models, particularly those based on generative architectures, for detecting anomalies in multivariate time series in an industrial setting. We successfully implemented and applied novel models on publicly available and real-world datasets, demonstrating their superiority over traditional and SOTA models. This chapter concludes the work and provides a brief overview of the main findings.

Initially, we formulated four challenges in industrial time series anomaly detection (cf. Section 1.2):

RQ1: How to overcome the scarcity of labeled anomaly data?

RQ2: Can generative models be used for time series anomaly detection?

RQ3: Can they outperform traditional models and techniques?

RQ4: How can they be integrated into smart manufacturing pipelines?

While labeling real-world data by experts is theoretically possible, it is time-consuming and would yield an unbalanced two-class dataset, as anomalies are rare events. The absence of labels during training (RQ1), as in publicly available datasets for time series

anomaly detection (cf. Chapter 6) and real-world data (cf. Chapter 7), requires model designs for unsupervised learning. Our focus lies on reconstruction-based approaches for time series data. These models do not need labels during training to optimize an objective.

In recent years, generative models have gained popularity within the deep learning community. While other domains, such as NLP and image generation, have been well-examined, tasks related to time series data, especially anomaly detection in real-world scenarios, remain mostly underexplored. Existing modern deep learning models, such as those based on the Transformer architecture, often rely on additional post-processing steps that adjust standardized metrics in a favorable manner. Working with these models as standalone solutions, on the other hand, results in poor detection scores due to poor reconstructions.

We have developed multiple novel models that detect anomalies in various scenarios in real time. The models, introduced in Chapter 4, are built on GANs, Transformers, and DDPMs, and are conditioned on the input signal itself or on extracted features, enabling them to accurately reconstruct the multivariate input. By utilizing standardized metrics (cf. Chapter 5) for thresholding, anomalies can be localized accurately and the performance of different architectures can be compared. The reconstruction and detection capabilities of the models presented in this work across different datasets were demonstrated in Chapters 6 and 7 (RQ2).

We tested straightforward techniques for time series anomaly detection on publicly available datasets. A simple z-normalization and PCA-Error calculations are unbeatably fast on and can detect in particular point anomalies. However, they require non-stationary time series data and a global representation of nominal states when their parameters are calculated. Further, we tested three SOTA algorithms for time series anomaly detection that are based on deep-learning architectures. When performing experiments on various input window sizes with standardized evaluations, they fail to accurately reconstruct the input signal and therefore yield low detection scores. Our transformer-based model Swin1D-AD shows the most stable performance across window sizes, datasets, and evaluation strategies. It outperforms both traditional and modern algorithms for anomaly detection in the experiments shown and can accurately reconstruct input signals in real time (RQ3).

Our modular implementation allows for building, training, and evaluating diverse architectures with different hyperparameter configurations in a local environment. By integrating the MLflow library into our framework, we are able to monitor and track relevant metrics, thereby making results reproducible. This approach also enables us to leverage MLflow's capabilities for efficient model versioning, storage, and deployment, which promotes the integration of our generative models into smart manufacturing pipelines (RQ4), where scalability, traceability, and model management are essential. As demon-

strated in the experiments conducted, our models are designed to handle varying input window sizes and achieve low inference latency for reconstructing inputs, making them well-suited for real-world scenarios that involve high-frequency data. Deviations between reconstructed and actual sensor values can trigger alarms based on application-specific thresholds, informing machine operators to investigate those unexpected sequences and monitor machine behavior further. In a fully automated setting, the system could autonomously reduce machine speed, halt production, or notify suppliers when a component fault is detected. The specific sensitivity levels for such responses, however, must be defined individually by each manufacturer.

8.1 Limitations

Within this work, we showed how to overcome common challenges regarding data and model structures. However, some limitations still exist. On the one hand, publicly available datasets are relatively short and may not fully represent the variability of real industrial or sensor data. On the other hand, real machines are complex to monitor due to the amount of data that must be recorded and stored. Their longevity can also involve the absence of anomalies during a recording period. Synthetic outliers used for benchmarking may not capture all real-world anomaly types and patterns.

The models presented in this work are designed to be scalable with respect to input sizes and easily modifiable in their internal structure. However, the hyperparameter tuning and architecture optimization were constrained by available computational resources.

The different models yield the anomaly-free reconstruction of the input signal. The reconstruction can therefore be seen as what the model expects the input to look like based on the features or condition. By displaying both signals, the resulting decision about whether an anomaly is present becomes interpretable to a user. However, it remains challenging to understand the reasons behind the model's reconstruction of the time series. This lack of transparency might hinder adoption in safety-critical or regulated industries.

8.2 Outlook of Time Series Anomaly Detection

The length of the input context window is crucial for detecting and generating patterns in time series data. While publicly available datasets are relatively short, as depicted in Chapter 6, this hyperparameter is not often addressed in current literature about time series anomaly detection. We demonstrated the necessity of large input windows in real-world applications when inter-channel correlations exist and inference time is critical. Especially with the KIWA dataset, we reconstructed acceleration data from a machine and can thereby detect anomalies.

However, a different area of deep learning models that benefit from large context windows is text understanding and generation. Recent models, such as SmolLM3 [192], can process up to 128,000 tokens and are used to analyze unstructured data, like documents. The context size is much larger than that introduced in this work, but it also comes with higher training costs. While we used a single consumer Nvidia RTX 3080Ti GPU to train our models for a limited time (minutes to hours), the authors of SmolLM3 used 384 Nvidia H100 GPUs for 24 days.

The development of LLMs also have a positive effect on different domains, such as time series analysis, as they push the boundaries of processing capabilities. LLMs are often based on transformer-like architectures, which can be used for anomaly detection, as we have done with Swin1D-AD. Over the past few months, several attempts have been made to implement foundation models for time series data, mainly for the purpose of forecasting univariate data [193, 194, 195]. Their primary goal is to train the model once on various datasets to gain a general understanding of contextual concepts and normal data distribution. However, foundation models for time series data come along with additional challenges.

While the input data length can be specified globally for both text and time series data, datasets containing multivariate signals have different numbers of channels, depending on the sensors used. A possible solution was presented in [193], where the input data is converted from $\mathbb{R}^{T \times c}$ to $c \mathbb{R}^T$ data points. As each channel is treated separately, spatial relations that cause anomalies cannot be identified by design.

In an industrial setting, high sampling rates are common, and inference must be performed in real-time, often on-device. Foundation models with multiple billions of parameters are not suitable in this scenario due to their size and inference time.

8.3 Final Remarks

In summary, the models and analyses presented in this thesis have demonstrated that different types of models can effectively capture temporal and spatial dependencies and improve the detection of anomalies in multivariate time series. Their generative nature does not hinder their performance, and the deployment of modern layers enhances the ability to reconstruct complex patterns. This work has provided both methodological insights and practical contributions to the area of time series anomaly detection by systematically addressing model-specific disadvantages and selecting appropriate conditioning techniques.

The framework developed highlights the importance of architectural choices, data representation, and evaluation strategies when assessing model performance. Together, these findings provide a foundation for more transparent and reproducible research in anomaly

detection, which continues to gain importance as data-driven systems become more integrated into industrial domains.

While several challenges in time series anomaly detection remain, particularly concerning model interpretability, the framework and models presented in this work are a step towards robust and scalable architectures. By establishing AI and deep-learning-based generative models as the primary instruments for anomaly detection, the concepts presented here aim to further close the gap between academia and industry.

BIBLIOGRAPHY

- [1] Y. Lu, “Industry 4.0: A survey on technologies, applications and open research issues,” *Journal of Industrial Information Integration*, vol. 6, pp. 1–10, 2017.
- [2] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, “The industrial internet of things (iiot): An analysis framework,” *Computers in Industry*, vol. 101, pp. 1–12, 2018.
- [3] M. Javaid, A. Haleem, R. P. Singh, S. Rab, and R. Suman, “Significance of sensors for industry 4.0: Roles, capabilities, and applications,” *Sensors International*, vol. 2, p. 100110, 2021.
- [4] A. Schütze, N. Helwig, and T. Schneider, “Sensors 4.0 – smart sensors and measurement technology enable industry 4.0,” *Journal of Sensors and Sensor Systems*, vol. 7, no. 1, pp. 359–371, 2018.
- [5] D. Zuehlke, “Smartfactory—towards a factory-of-things,” *Annual Reviews in Control*, vol. 34, no. 1, pp. 129–138, 2010.
- [6] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM Comput. Surv.*, vol. 41, no. 3, Jul. 2009.
- [7] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, “A review on outlier/anomaly detection in time series data,” *ACM Comput. Surv.*, vol. 54, no. 3, Apr. 2021.
- [8] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, “Lstm-based encoder-decoder for multi-sensor anomaly detection,” in *ICML Workshop on Anomaly Detection*, 2016.
- [9] J. D. Hamilton, *Time Series Analysis*. Princeton University Press, 1994.
- [10] Z. Li, Y. Zhao, J. Han, Y. Su, R. Jiao, X. Wen, and D. Pei, “Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3220–3230.
- [11] Y. Dodge, *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 536–539.

- [12] A. C. Harvey and S. Peters, “Estimation procedures for structural time series models,” *Journal of Forecasting*, vol. 9, no. 2, p. 89–108, Mar. 1990.
- [13] K.-H. Lai, D. Zha, J. Xu, and Y. Zhao, “Revisiting time series outlier detection: Definitions and benchmarks,” in *NeurIPS Datasets and Benchmarks*, 2021.
- [14] M. Ahmed, A. Naser Mahmood, and J. Hu, “A survey of network anomaly detection techniques,” *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.
- [15] Y. Li, N. Liu, J. Li, M. Du, and X. Hu, “Deep structured cross-modal anomaly detection,” *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019.
- [16] Z. Zamanzadeh Darban, G. I. Webb, S. Pan, C. Aggarwal, and M. Salehi, “Deep learning for time series anomaly detection: A survey,” *ACM Computing Surveys*, vol. 57, no. 1, Oct. 2024.
- [17] M. Müller, *Dynamic Time Warping*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 69–84.
- [18] W. Choi, J. Cho, S. Lee, and Y. Jung, “Fast constrained dynamic time warping for similarity measure of time series data,” *IEEE Access*, vol. 8, pp. 222 841–222 858, 2020.
- [19] Q. Zhang, C. Zhang, L. Cui, X. Han, Y. Jin, G. Xiang, and Y. Shi, “A method for measuring similarity of time series based on series decomposition and dynamic time warping,” *Applied Intelligence*, vol. 53, no. 6, p. 6448–6463, Jul. 2022.
- [20] A. Geiger, D. Liu, S. Alnegheimish, A. Cuesta-Infante, and K. Veeramachaneni, “Tadgan: Time series anomaly detection using generative adversarial networks,” *CoRR*, vol. abs/2009.07769, 2020.
- [21] M. Salehi, H. Mirzaei, D. Hendrycks, Y. Li, M. H. Rohban, and M. Sabokrou, “A unified survey on anomaly, novelty, open-set, and out-of-distribution detection: Solutions and future challenges,” *Transactions on Machine Learning Research*, vol. 2022, 2021.
- [22] J. Yang, K. Zhou, Y. Li, and Z. Liu, “Generalized out-of-distribution detection: A survey,” *International Journal of Computer Vision*, vol. 132, no. 12, p. 5635–5662, Jun. 2024.
- [23] F. E. Grubbs, “Procedures for detecting outlying observations in samples,” *Technometrics*, vol. 11, no. 1, pp. 1–21, 1969.

- [24] D. M. Hawkins, *Identification of Outliers*. Dordrecht: Springer Netherlands, 1980, pp. 1–12.
- [25] V. Hodge and J. Austin, “A survey of outlier detection methodologies,” *Artificial Intelligence Review*, vol. 22, no. 2, p. 85–126, Oct. 2004.
- [26] X. Li, Z. Fang, Y. Zhang, N. Ma, J. Bu, B. Han, and H. Wang, “Characterizing sub-manifold region for out-of-distribution detection,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 37, no. 1, pp. 130–147, 2025.
- [27] D. Hendrycks and K. Gimpel, “A baseline for detecting misclassified and out-of-distribution examples in neural networks,” in *International Conference on Learning Representations*, 2017.
- [28] S. Agrahari, S. Srivastava, and A. K. Singh, “Review on novelty detection in the non-stationary environment,” *Knowledge and Information Systems*, vol. 66, no. 3, p. 1549–1574, Nov. 2023.
- [29] H. Xu, S. Xu, and W. Yang, “Unsupervised industrial anomaly detection with diffusion models,” *Journal of Visual Communication and Image Representation*, vol. 97, p. 103983, 2023.
- [30] X. Yao, J. Zhou, J. Zhang, and C. R. Boër, “From intelligent manufacturing to smart manufacturing for industry 4.0 driven by next generation artificial intelligence and further on,” in *2017 5th International Conference on Enterprise Systems (ES)*, 2017, pp. 311–318.
- [31] D. Rothman, *Transformers for natural language processing and computer vision*, 3rd ed. Birmingham, England: Packt Publishing, Mar. 2024.
- [32] C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*. Springer International Publishing, 2024.
- [33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [34] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, “A review on generative adversarial networks: Algorithms, theory, and applications,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3313–3332, 2023.
- [35] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014.

- [36] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS’16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 2234–2242.
- [37] A. Aggarwal, M. Mittal, and G. Battineni, “Generative adversarial network: An overview of theory and applications,” *International Journal of Information Management Data Insights*, vol. 1, no. 1, p. 100004, 2021.
- [38] A. Madani, M. Moradi, A. Karargyris, and T. Syeda-Mahmood, “Semi-supervised learning with generative adversarial networks for chest x-ray classification with ability of data domain adaptation,” in *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, 2018, pp. 1038–1042.
- [39] M. D. Cirillo, D. Abramian, and A. Eklund, “Vox2vox: 3d-gan for brain tumour segmentation,” in *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, A. Crimi and S. Bakas, Eds. Cham: Springer International Publishing, 2021, pp. 274–284.
- [40] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS’16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 2234–2242.
- [41] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. The MIT Press, 1986.
- [42] H. Bourlard and Y. Kamp, “Auto-association by multilayer perceptrons and singular value decomposition,” *Biological Cybernetics*, vol. 59, no. 4–5, p. 291–294, Sep. 1988.
- [43] M. a. Ranzato, C. Poultney, S. Chopra, and Y. Cun, “Efficient learning of sparse representations with an energy-based model,” in *Advances in Neural Information Processing Systems*, B. Schölkopf, J. Platt, and T. Hoffman, Eds., vol. 19. MIT Press, 2006, pp. 1137–1144.
- [44] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML ’08. New York, NY, USA: Association for Computing Machinery, 2008, p. 1096–1103.
- [45] G. Alain and Y. Bengio, “What regularized auto-encoders learn from the data-generating distribution,” *Journal of Machine Learning Research*, vol. 15, no. 1, p. 3563–3593, Jan. 2014.

- [46] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.
- [47] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-Image Translation with Conditional Adversarial Networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2017, pp. 5967–5976.
- [48] J. Ho, A. Jain, and P. Abbeel, “Denoising diffusion probabilistic models,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 6840–6851.
- [49] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” 2013.
- [50] D. J. Rezende, S. Mohamed, and D. Wierstra, “Stochastic backpropagation and approximate inference in deep generative models,” in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 2. Beijing, China: PMLR, 22–24 Jun 2014, pp. 1278–1286.
- [51] D. M. Blei, A. Kucukelbir, and J. D. M. and, “Variational inference: A review for statisticians,” *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.
- [52] S. Kullback and R. A. Leibler, “On information and sufficiency,” *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.
- [53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *CoRR*, vol. abs/1706.03762, 2017.
- [54] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *CoRR*, vol. abs/1409.0473, 2014.
- [55] N. J. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, “Image transformer,” in *International Conference on Machine Learning (ICML)*, 2018.
- [56] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *International Conference on Learning Representations*, 2021.

- [57] R. Luo, L. Sun, Y. Xia, T. Qin, S. Zhang, H. Poon, and T.-Y. Liu, “Biogpt: generative pre-trained transformer for biomedical text generation and mining,” *Briefings in Bioinformatics*, vol. 23, no. 6, 09 2022.
- [58] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, “Deep unsupervised learning using nonequilibrium thermodynamics,” in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 2256–2265.
- [59] C. Luo, “Understanding diffusion models: A unified perspective,” 2022.
- [60] S. Addepalli, G. K. Nayak, A. Chakraborty, and V. B. Radhakrishnan, “Degan: Data-enriching gan for retrieving representative samples from a trained classifier,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, p. 3130–3137, Apr. 2020.
- [61] D. Parthasarathy, K. Backstrom, J. Henriksson, and S. Einarsdottir, “Controlled time series generation for automotive software-in-the-loop testing using GANs,” in *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2020, pp. 39–46.
- [62] M. Arjovsky and L. Bottou, “Towards principled methods for training generative adversarial networks,” in *International Conference on Learning Representations*, 2017.
- [63] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *CoRR*, vol. abs/1511.06434, 2015.
- [64] T. Karras, S. Laine, and T. Aila, “A style-based generator architecture for generative adversarial networks,” *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 43, no. 12, pp. 4217–4228, dec 2021.
- [65] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 214–223.
- [66] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, “Improved training of wasserstein gans,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS’17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 5769–5779.

- [67] K. Berahmand, F. Daneshfar, E. S. Salehi, Y. Li, and Y. Xu, “Autoencoders and their applications in machine learning: a survey,” *Artificial Intelligence Review*, vol. 57, no. 2, Feb. 2024.
- [68] C. Baur, S. Denner, B. Wiestler, N. Navab, and S. Albarqouni, “Autoencoders for unsupervised anomaly segmentation in brain mr images: A comparative study,” *Medical Image Analysis*, vol. 69, p. 101952, 2021.
- [69] G. A. Pereira and M. Hussain, “A review of transformer-based models for computer vision tasks: Capturing global context and spatial relationships,” *ArXiv*, vol. abs/2408.15178, 2024.
- [70] P. Dhariwal and A. Q. Nichol, “Diffusion models beat GANs on image synthesis,” in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021.
- [71] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, and M.-H. Yang, “Diffusion models: A comprehensive survey of methods and applications,” *ACM Comput. Surv.*, vol. 56, no. 4, Nov. 2023.
- [72] S. Schmidl, P. Wenig, and T. Papenbrock, “Anomaly detection in time series: a comprehensive evaluation,” *Proceedings of the VLDB Endowment*, vol. 15, no. 9, p. 1779–1797, may 2022.
- [73] B. Kim, M. A. Alawami, E. Kim, S. Oh, J. Park, and H. Kim, “A comparative study of time series anomaly detection models for industrial control systems,” *Sensors*, vol. 23, no. 3, p. 1310, Jan. 2023.
- [74] C. C. Aggarwal, *Outlier Analysis*. Cham: Springer International Publishing, 2017, pp. 1–34.
- [75] T. Kim, J. Kim, Y. Tae, C. Park, J.-H. Choi, and J. Choo, “Reversible instance normalization for accurate time-series forecasting against distribution shift,” in *International Conference on Learning Representations*, 2021.
- [76] H. Steinhaus, “Sur la division des corps matériels en parties,” *Bulletin de l'Académie Polonaise des Sciences*, vol. Cl. III — Vol. IV, no. 12, pp. 801–804, 1956.
- [77] Z. Cebeci, C. Cebeci, Y. Tahtali, and L. Bayyurt, “Two novel outlier detection approaches based on unsupervised possibilistic and fuzzy clustering,” *PeerJ Computer Science*, vol. 8, p. e1060, Sep. 2022.
- [78] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “Lof: identifying density-based local outliers,” in *Proceedings of the 2000 ACM SIGMOD International Conference*

- on *Management of Data*, ser. SIGMOD '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 93–104.
- [79] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.
 - [80] R. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 3rd ed. Australia: OTexts, 2021.
 - [81] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis*. Wiley, Jun. 2008.
 - [82] P. R. Winters, “Forecasting sales by exponentially weighted moving averages,” *Management Science*, vol. 6, no. 3, pp. 324–342, 1960.
 - [83] K. P. and, “Liii. on lines and planes of closest fit to systems of points in space,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.
 - [84] Q. Wen, J. Gao, X. Song, L. Sun, H. Xu, and S. Zhu, “RobustSTL: a robust seasonal-trend decomposition algorithm for long time series,” in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.
 - [85] X. He, Y. Li, J. Tan, B. Wu, and F. Li, “OneShotSTL: One-shot seasonal-trend decomposition for online time series anomaly detection and forecasting,” *Proceedings of the VLDB Endowment*, vol. 16, no. 6, p. 1399–1412, Feb. 2023.
 - [86] J. Zhang, F.-C. Tsui, M. M. Wagner, and W. R. Hogan, “Detection of outbreaks from time series data using wavelet transform,” *AMIA Annual Symposium proceedings. AMIA Symposium*, pp. 748–52, 2003.
 - [87] Y. Wang, H. Wu, J. Dong, Y. Liu, M. Long, and J. Wang, “Deep time series models: A comprehensive survey and benchmark,” 2024.
 - [88] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals,

- P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [89] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” 2017.
- [90] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [91] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder–decoder approaches,” in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, D. Wu, M. Carpuat, X. Carreras, and E. M. Vecchi, Eds. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 103–111.
- [92] 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*, ser. KDD ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2828–2837.
- [93] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, “Unsupervised anomaly detection with generative adversarial networks to guide marker discovery,” 2017.
- [94] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *CoRR*, vol. abs/1511.06434, 2015.
- [95] H. Zenati, C.-S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, “Efficient gan-based anomaly detection,” *ArXiv*, vol. abs/1802.06222, 2018.
- [96] J. Donahue, P. Krähenbühl, and T. Darrell, “Adversarial feature learning,” in *International Conference on Learning Representations*, 2017.
- [97] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, “Ganomaly: Semi-supervised anomaly detection via adversarial training,” in *Computer Vision – ACCV 2018*, C. V. Jawahar, H. Li, G. Mori, and K. Schindler, Eds. Cham: Springer International Publishing, 2019, pp. 622–637.
- [98] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, “Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks,” in *Artificial*

- Neural Networks and Machine Learning – ICANN 2019: Text and Time Series*, I. V. Tetko, V. Kůrková, P. Karpov, and F. Theis, Eds. Cham: Springer International Publishing, 2019, pp. 703–716.
- [99] 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*. New York, NY, USA: Association for Computing Machinery, 2020, p. 3395–3404.
- [100] H. Wu, J. Xu, J. Wang, and M. Long, “Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting,” in *Neural Information Processing Systems*, 2021.
- [101] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, “FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting,” in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 27 268–27 286.
- [102] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, “A time series is worth 64 words: Long-term forecasting with transformers,” 2023.
- [103] S. Tuli, G. Casale, and N. R. Jennings, “Tranad: Deep transformer networks for anomaly detection in multivariate time series data,” *Proc. VLDB Endow.*, vol. 15, no. 6, p. 1201–1214, feb 2022.
- [104] 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*, ser. NIPS ’23. Red Hook, NY, USA: Curran Associates Inc., 2024.
- [105] J. Xu, H. Wu, J. Wang, and M. Long, “Anomaly transformer: Time series anomaly detection with association discrepancy,” in *International Conference on Learning Representations*, 2022.
- [106] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, “itransformer: Inverted transformers are effective for time series forecasting,” in *The Twelfth International Conference on Learning Representations*, 2024.
- [107] D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach, “SDXL: Improving latent diffusion models for high-resolution image synthesis,” in *The Twelfth International Conference on Learning Representations*, 2024.

- [108] A. Q. Nichol and P. Dhariwal, “Improved denoising diffusion probabilistic models,” in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8162–8171.
- [109] M. Kollovieh, A. F. Ansari, M. Bohlke-Schneider, J. Zschiegner, H. Wang, and Y. B. Wang, “Predict, refine, synthesize: Self-guiding diffusion models for probabilistic time series forecasting,” in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 28 341–28 364.
- [110] K. Rasul, C. Seward, I. Schuster, and R. Vollgraf, “Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting,” in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8857–8868.
- [111] X. Fan, Y. Wu, C. Xu, Y. Huang, W. Liu, and J. Bian, “MG-TSD: Multi-granularity time series diffusion models with guided learning process,” in *The Twelfth International Conference on Learning Representations*, 2024.
- [112] C. Xiao, Z. Gou, W. Tai, K. Zhang, and F. Zhou, “Imputation-based time-series anomaly detection with conditional weight-incremental diffusion models,” in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD ’23. New York, NY, USA: Association for Computing Machinery, 2023, p. 2742–2751.
- [113] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.
- [114] Y. Chen, C. Zhang, M. Ma, Y. Liu, R. Ding, B. Li, S. He, S. Rajmohan, Q. Lin, and D. Zhang, “ImDiffusion: Imputed diffusion models for multivariate time series anomaly detection,” *Proc. VLDB Endow.*, vol. 17, no. 3, pp. 359–372, November 2023.
- [115] I. Pintilie, A. Manolache, and F. Brad, “Time Series Anomaly Detection using Diffusion-based Models,” in *2023 IEEE International Conference on Data Mining Workshops (ICDMW)*. Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2023, pp. 570–578.

- [116] C. Yang, T. Wang, and X. Yan, “DDMT: Denoising diffusion mask transformer models for multivariate time series anomaly detection,” *ArXiv*, vol. abs/2310.08800, 2023.
- [117] S. Li and J. Wen, “A model-based fault detection and diagnostic methodology based on pca method and wavelet transform,” *Energy and Buildings*, vol. 68, pp. 63–71, 2014.
- [118] X. Zhang, N. Li, J. Li, T. Dai, Y. Jiang, and S. Xia, “Unsupervised surface anomaly detection with diffusion probabilistic model,” in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 6759–6768.
- [119] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, “Informer: Beyond efficient transformer for long sequence time-series forecasting,” in *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference*, vol. 35, no. 12. AAAI Press, 2021, pp. 11 106–11 115.
- [120] W. Zhang, H. Wang, and F. Zhang, “Skip-timeformer: skip-time interaction transformer for long sequence time-series forecasting,” in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, ser. IJCAI ’24, 2024.
- [121] A. Chen, A. Chow, A. Davidson, A. DCunha, A. Ghodsi, S. A. Hong, A. Konwinski, C. Mewald, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, A. Singh, F. Xie, M. Zaharia, R. Zang, J. Zheng, and C. Zumar, “Developments in mlflow: A system to accelerate the machine learning lifecycle,” in *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*, ser. DEEM ’20. New York, NY, USA: Association for Computing Machinery, 2020.
- [122] A. Abid, A. Abdalla, A. Abid, D. Khan, A. Alfozan, and J. Y. Zou, “Gradio: Hassle-free sharing and testing of ML models in the wild,” *CoRR*, vol. abs/1906.02569, 2019.
- [123] M. Hoh, A. Schöttl, H. Schaub, and F. Wenninger, “A generative model for anomaly detection in time series data,” *Procedia Computer Science*, vol. 200, p. 629–637, 2022.
- [124] M. Hoh, A. Schöttl, H. Schaub, and N. Leuze, “Generative anomaly detection in multivariate time series,” in *Proceedings of the 3rd Winter IFSA Conference on Automation, Robotics & Communications for Industry 4.0/5.0*. IFSA Publishing, 2023, p. 171–174.
- [125] C. Villani, *The Wasserstein distances*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 93–111.

- [126] N.-T. Tran, T.-A. Bui, and N.-M. Cheung, “Dist-gan: An improved gan using distance constraints,” in *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8–14, 2018, Proceedings, Part XIV*. Berlin, Heidelberg: Springer-Verlag, 2018, p. 387–401.
- [127] A. Ng, “Cs294a lecture notes: Sparse autoencoder,” pp. 1–19, 2011.
- [128] K. Zhou, S. Gao, J. Cheng, Z. Gu, H. Fu, Z. Tu, J. Yang, Y. Zhao, and J. Liu, “Sparse-gan: Sparsity-constrained generative adversarial network for anomaly detection in retinal oct image,” *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*, pp. 1227–1231, 2019.
- [129] J. T. Zhou, K. Di, J. Du, X. Peng, H. Yang, S. J. Pan, I. Tsang, Y. Liu, Z. Qin, and R. S. M. Goh, “Sc2net: Sparse lstms for sparse coding,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018.
- [130] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *International Conference on Learning Representations*, 2019.
- [131] C.-C. Chiu and C. Raffel, “Monotonic chunkwise attention,” in *International Conference on Learning Representations*, 2018.
- [132] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin transformer: Hierarchical vision transformer using shifted windows,” in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9992–10 002.
- [133] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo, “Swin transformer v2: Scaling up capacity and resolution,” in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 11 999–12 009.
- [134] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan, *Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting*. Red Hook, NY, USA: Curran Associates Inc., 2019.
- [135] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, “Transformers are rnns: fast autoregressive transformers with linear attention,” in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML’20. JMLR.org, 2020.
- [136] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” 2016.
- [137] J. Yang, C. Li, P. Zhang, X. Dai, B. Xiao, L. Yuan, and J. Gao, “Focal attention for long-range interactions in vision transformers,” in *Advances in Neural Information*

- Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 30 008–30 022.
- [138] A. Hassani and H. Shi, “Dilated neighborhood attention transformer,” 2022.
 - [139] I. Beltagy, M. E. Peters, and A. Cohan, “Longformer: The long-document transformer,” *arXiv:2004.05150*, 2020.
 - [140] J. Yuan, H. Gao, D. Dai, J. Luo, L. Zhao, Z. Zhang, Z. Xie, Y. X. Wei, L. Wang, Z. Xiao, Y. Wang, C. Ruan, M. Zhang, W. Liang, and W. Zeng, “Native sparse attention: Hardware-aligned and natively trainable sparse attention,” 2025.
 - [141] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, “Flashattention: fast and memory-efficient exact attention with io-awareness,” in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS ’22. Red Hook, NY, USA: Curran Associates Inc., 2022.
 - [142] T. Dao, “Flashattention-2: Faster attention with better parallelism and work partitioning,” in *The Twelfth International Conference on Learning Representations*, 2024.
 - [143] B. He and T. Hofmann, “Simplifying transformer blocks,” in *The Twelfth International Conference on Learning Representations*, 2024.
 - [144] J. Zhu, X. Chen, K. He, Y. LeCun, and Z. Liu, “Transformers without normalization,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
 - [145] M. Hoh, H. Schaub, N. Leuze, and A. Schöttl, “Rethinking time series anomaly detection: A scalable transformer-based framework for large contexts,” in *2025 11th International Conference on Control, Automation and Robotics (ICCAR)*, 2025, pp. 295–302.
 - [146] H. Cao, Y. Wang, J. Chen, D. Jiang, X. Zhang, Q. Tian, and M. Wang, “Swin-unet: Unet-like pure transformer for medical image segmentation,” in *Computer Vision – ECCV 2022 Workshops*, L. Karlinsky, T. Michaeli, and K. Nishino, Eds. Cham: Springer Nature Switzerland, 2023, pp. 205–218.
 - [147] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, “Score-based generative modeling through stochastic differential equations,” in *International Conference on Learning Representations*, 2021.
 - [148] A. Jabri, D. J. Fleet, and T. Chen, “Scalable adaptive computation for iterative generation,” in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML’23. JMLR.org, 2023.

- [149] S. Lee, K. Lee, and T. Park, “ANT: Adaptive noise schedule for time series diffusion models,” in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
- [150] T. Chen, “On the importance of noise scheduling for diffusion models,” 2023.
- [151] E. Hoogeboom, J. Heek, and T. Salimans, “Simple diffusion: end-to-end diffusion for high resolution images,” in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML’23. JMLR.org, 2023.
- [152] M. Kim, D. Ki, S.-W. Shim, and B.-J. Lee, “Adaptive Non-Uniform Timestep Sampling for Accelerating Diffusion Model Training,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2025, pp. 2513–2522.
- [153] H. Jiang, M. Imran, T. Zhang, Y. Zhou, M. Liang, K. Gong, and W. Shao, “Fast-ddpm: Fast denoising diffusion probabilistic models for medical image-to-image generation,” *IEEE Journal of Biomedical and Health Informatics*, 2025.
- [154] T. Hang, S. Gu, C. Li, J. Bao, D. Chen, H. Hu, X. Geng, and B. Guo, “Efficient diffusion training via min-snr weighting strategy,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023, pp. 7441–7451.
- [155] A. Bansal, E. Borgnia, H.-M. Chu, J. Li, H. Kazemi, F. Huang, M. Goldblum, J. Geiping, and T. Goldstein, “Cold diffusion: Inverting arbitrary image transforms without noise,” in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 41 259–41 282.
- [156] J. Song, C. Meng, and S. Ermon, “Denoising diffusion implicit models,” in *International Conference on Learning Representations*, 2021.
- [157] F. Behrendt, D. Bhattacharya, R. Mieling, L. Maack, J. Krüger, R. Opfer, and A. Schlaefer, “Guided reconstruction with conditioned diffusion models for unsupervised anomaly detection in brain mris,” *Computers in Biology and Medicine*, vol. 186, p. 109660, 2025.
- [158] T. Chen, R. Zhang, and G. E. Hinton, “Analog bits: Generating discrete data using diffusion models with self-conditioning,” *ArXiv*, vol. abs/2208.04202, 2022.
- [159] C. Saharia, J. Ho, W. Chan, T. Salimans, D. J. Fleet, and M. Norouzi, “Image super-resolution via iterative refinement,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, pp. 4713–4726, 2023.

- [160] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. Courville, “Film: visual reasoning with a general conditioning layer,” in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI’18/IAAI’18/EAAI’18. AAAI Press, 2018.
- [161] Z. Ma, Y. Zhang, G. Jia, L. Zhao, Y. Ma, M. Ma, G. Liu, K. Zhang, N. Ding, J. Li, and B. Zhou, “Efficient Diffusion Models: A Comprehensive Survey From Principles to Practices,” *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 47, no. 09, pp. 7506–7525, Sep. 2025.
- [162] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.
- [163] Z. Zhuang, M. Liu, A. Cutkosky, and F. Orabona, “Understanding adamw through proximal methods and scale-freeness,” *Transactions on Machine Learning Research*, 2022.
- [164] H. Kriegel, P. Kröger, E. Schubert, and A. Zimek, “Interpreting and unifying outlier scores,” in *Proceedings of the 11th SIAM International Conference on Data Mining, SDM 2011*, B. Liu, H. Liu, C. Clifton, T. Washio, and C. Kamath, Eds. United States: Society for Industrial and Applied Mathematics, Dec. 2011, pp. 13–24, 11th SIAM International Conference on Data Mining, SDM.
- [165] F. Iglesias Vázquez, H. O. Marques, A. Zimek, and T. Zseby, “What do anomaly scores actually mean? dynamic characteristics beyond accuracy,” *Data Mining and Knowledge Discovery*, vol. 39, no. 1, Nov. 2024.
- [166] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet, “Anomaly detection in streams with extreme value theory,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1067–1075.
- [167] 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, 2021.
- [168] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, and H. Qiao, “Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications,” in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW ’18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, p. 187–196.

- [169] S. Kim, K. Choi, H.-S. Choi, B. Lee, and S. Yoon, “Towards a rigorous evaluation of time-series anomaly detection,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 7, pp. 7194–7201, Jun. 2022.
- [170] D. Wagner, T. Michels, F. C. Schulz, A. Nair, M. Rudolph, and M. Kloft, “TimeseAD: Benchmarking deep multivariate time-series anomaly detection,” *Transactions on Machine Learning Research*, 2023.
- [171] J. Davis and M. Goadrich, “The relationship between precision-recall and roc curves,” in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML ’06. New York, NY, USA: Association for Computing Machinery, 2006, p. 233–240.
- [172] T. Saito and M. Rehmsmeier, “The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets,” *PLOS ONE*, vol. 10, no. 3, p. e0118432, Mar. 2015.
- [173] J. Paparrizos, P. Boniol, T. Palpanas, R. S. Tsay, A. Elmore, and M. J. Franklin, “Volume under the surface: a new accuracy evaluation measure for time-series anomaly detection,” *Proc. VLDB Endow.*, vol. 15, no. 11, p. 2774–2787, Jul. 2022.
- [174] C. Wang, Z. Zhuang, Q. Qi, J. Wang, X. Wang, H. Sun, and J. Liao, “Drift doesn’t matter: Dynamic decomposition with diffusion reconstruction for unstable multivariate time series anomaly detection,” in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 10 758–10 774.
- [175] M. El Amine Sehili and Z. Zhang, “Multivariate time series anomaly detection: Fancy algorithms and flawed evaluation methodology,” in *Performance Evaluation and Benchmarking*, R. Nambiar and M. Poess, Eds. Cham: Springer Nature Switzerland, 2024, pp. 1–17.
- [176] M. S. Sarfraz, M.-Y. Chen, L. Layer, K. Peng, and M. Koulakis, “Position: Quo vadis, unsupervised time series anomaly detection?” in *Proceedings of the 41st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds., vol. 235. PMLR, 21–27 Jul 2024, pp. 43 461–43 476.
- [177] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Söderström, “Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding,” *CoRR*, 2018.

- [178] J. Kang, Y. Shin, and J.-G. Lee, “Vardrop: Enhancing training efficiency by reducing variate redundancy in periodic time series forecasting,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 17, pp. 17 742–17 749, Apr. 2025.
- [179] D. Entekhabi, S. Yueh, and P. E. O’Neill, *SMAP Handbook: Mapping Soil Moisture and Freeze/Thaw from Space*, Pasadena, CA, 2014, available at https://asf.alaska.edu/wp-content/uploads/2019/03/smap_handbook.pdf.
- [180] “Mars science laboratory/curiosity,” National Aeronautics and Space Administration (NASA), Tech. Rep., 2012. [Online]. Available: <https://science.nasa.gov/wp-content/uploads/2023/09/mars-science-laboratory.pdf>
- [181] J. L. Rodgers and W. A. Nicewander, “Thirteen ways to look at the correlation coefficient,” *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.
- [182] A. Abdulaal, Z. Liu, and T. Lancewicki, “Practical approach to asynchronous multivariate time series anomaly detection and localization,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2485–2494.
- [183] A. P. Mathur and N. O. Tippenhauer, “Swat: a water treatment testbed for research and training on ics security,” in *2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*, 2016, pp. 31–36.
- [184] J. Goh, S. Adepur, K. N. Junejo, and A. Mathur, “A dataset to support research in the design of secure water treatment systems,” in *Critical Information Infrastructures Security*. Cham: Springer International Publishing, 2017, pp. 88–99.
- [185] C. M. Ahmed, V. R. Palleti, and A. P. Mathur, “Wadi: a water distribution testbed for research in the design of secure cyber physical systems,” in *Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks*, ser. CySWATER ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 25–28.
- [186] R. Wu and E. J. Keogh, “Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress,” *IEEE Transactions on Knowledge & Data Engineering*, vol. 35, no. 03, pp. 2421–2429, mar 2023.
- [187] A. Deng and B. Hooi, “Graph neural network-based anomaly detection in multivariate time series,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 4027–4035.

- [188] N. Dogariu and M. Dogariu, “Apparatus and method for transferring electronic components from a first carrier to a second carrier,” Patent DE102011017218B4. [Online]. Available: <https://patents.google.com/patent/DE102011017218B4>
- [189] U. Prakapenka and D. R. Miehlich, “Apparatus and method for positioning an electronic component and/or a carrier relative to an ejector,” Patent DE102011104225B4. [Online]. Available: <https://patents.google.com/patent/DE102011104225B4>
- [190] G. Schiller, “Positionierungsvorrichtung und verfahren für die Übertragung elektronischer bauteile,” Patent EP1678744A1. [Online]. Available: <https://patents.google.com/patent/EP1678744A1>
- [191] *Seal Monitoring Lösungen*, Eagle Burgmann.
- [192] E. Bakouch, L. Ben Allal, A. Lozhkov, N. Tazi, L. Tunstall, C. M. Patiño, E. Beeching, A. Roucher, A. J. Reedi, Q. Gallouédec, K. Rasul, N. Habib, C. Fourier, H. Kydlicek, G. Penedo, H. Larcher, M. Morlon, V. Srivastav, J. Lochner, X.-S. Nguyen, C. Raffel, L. von Werra, and T. Wolf, “SmolLM3: smol, multilingual, long-context reasoner,” <https://huggingface.co/blog/smollm3>, 2025.
- [193] S. J. Talukder, Y. Yue, and G. Gkioxari, “TOTEM: Tokenized time series EMbeddings for general time series analysis,” *Transactions on Machine Learning Research*, 2024.
- [194] A. Das, W. Kong, R. Sen, and Y. Zhou, “A decoder-only foundation model for time-series forecasting,” 2024.
- [195] Y. Liu, G. Qin, Z. Shi, Z. Chen, C. Yang, X. Huang, J. Wang, and M. Long, “Sundial: A family of highly capable time series foundation models,” 2025.

CURRICULUM VITAE

The author contributed to the following publications. Although these deal with research areas outside the scope of this thesis, collaboration and exchange with colleagues have laid an academic foundation for many of the concepts presented here.

Doğan, S., Hoh, M., Leuze, N., R.-Peña, N., Schöttl, A. *Towards Solar Altitude Guided Scene Illumination*. IEEE Sensors 2025.

R.-Peña, N., Hoh, M., Doğan, S., Leuze, N., Schöttl, A. *Integrating Physical Effects into Action-Conditioned Frame Prediction for Robotic Manipulation*. International Congress on Human-Computer Interaction, Optimization and Robotic Applications (ICHORA) 2025.

Hoh, M., Schaub, H., Leuze, N., Schöttl, A. *Rethinking Time Series Anomaly Detection: A Scalable Transformer-Based Framework for Large Contexts*. International Conference on Control, Automation and Robotics (ICCAR) 2025.

Leuze, N., Schaub, H., Hoh, M., Doğan, S., R.-Peña, N., Voss, N., Schöttl, A. *Fostering Sparsity in Sparse Convolution Networks for Efficient 3D Perception via Feature-Guidance*. IEEE Sensors 2024.

Schaub, H., Wolff, C., Hoh, M., Schöttl, A. *Probabilistic Closed-Loop Active Grasping*. International Conference on Intelligent Robots and Systems (IROS) 2024.

Schaub, H., Leuze, N., Hoh, M., Schöttl, A. *Probabilistic Fusion of Depth Maps with Local Variance Estimation*. IEEE Sensors 2023.

Leuze, N., Schaub, H., Hoh, M., Schöttl, A. *SSL-VoxPart: A Novel Solid-State LiDAR-Tailored Voxel Partition Approach for 3D Perception*. IEEE Sensors 2023.

Hoh, M., Schöttl, A., Schaub, H., Leuze, N. *Generative Anomaly Detection in Multivariate Time Series*. Conference on Automation, Robotics & Communications for Industry 4.0/5.0/6.0 2023.

Schaub, H., Schöttl, A., Hoh, M. *Probabilistic Fusion of Depth Maps With a Reliable Estimation of the Local Reconstruction Quality*. IEEE Robotics and Automation Letters 2022.

Hoh, M., Schöttl, A., Schaub, H., Wenninger, F. *A Generative Model for Anomaly Detection in Time Series Data*. International Conference on Industry of the Future and Smart Manufacturing (ISM) 2022.

Schaub, H., Schöttl, A., Hoh, M. *6-DOF Grasp Detection for Unknown Objects Using Surface Reconstruction*. International Congress on Human-Computer Interaction, Optimization and Robotic Applications (ICHORA) 2021.



PUBLIC DATASETS DETAILS

This chapter provides supplementary material about the publicly available datasets. In particular, additional illustrations and more extensive tables are provided.

A.1 Tabular Per-Entity Overview

Tables A.1, A.2, and A.3 show the different entities of the used multi-entity datasets. Besides the entity name, the number of training and testing time steps is listed, and the percentage (%) of testing data that is labeled as abnormal. The **highest** and **lowest** values are highlighted by color.

A.2 Additional Dataset Plots

A.2.1 Value Ranges

Figure A.1 shows the value ranges of the presented publicly available datasets for each channel. The ranges of training and testing data are displayed side by side. Diamonds

Table A.1: Entities of the MSL dataset.

Entity	Training	Testing	%	Entity	Training	Testing	%
C-1	2,158	2,264	13.78	M-6	1,565	2,049	8.83
C-2	764	2,051	6.68	M-7	1,587	2,156	4.68
D-14	3,675	2,625	8.46	P-10	4,308	6,100	2.15
D-15	2,074	2,158	29.70	P-11	3,969	3,535	6.45
D-16	1,451	2,191	29.71	P-14	2,880	6,100	2.97
F-4	2,244	3,422	2.07	P-15	3,682	2,856	0.74
F-5	2,598	3,922	3.85	S-2	926	1,827	0.6
F-7	2,511	5,054	8.37	T-4	2,272	2,217	3.11
F-8	3,342	2,487	21.59	T-5	2,272	2,218	1.17
M-1	2,209	2,277	50.11	T-8	748	1519	6.71
M-2	2,208	2,277	50.11	T-9	439	1,096	10.22
M-3	2,037	2,127	11.80	T-12	1,145	2,430	4.98
M-4	2,076	2,038	12.32	T-13	1,145	2,430	10.37
M-5	2,032	2,303	13.07				

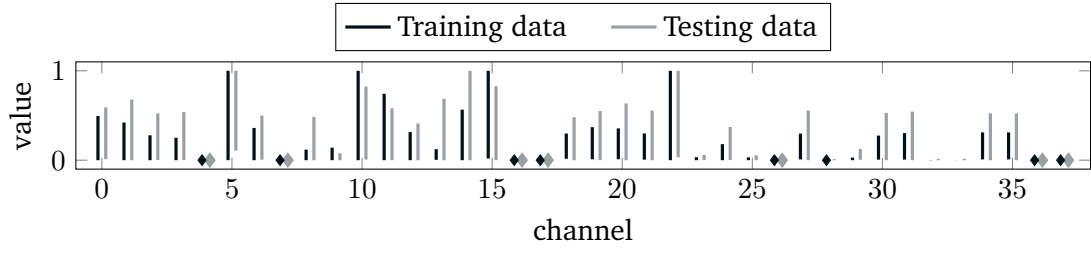
represent constant values across the whole time series. We omit MSL and SMD as they primarily consist of binary signals. For SMD, only one entity is presented.

A.2.2 Dataset Examples

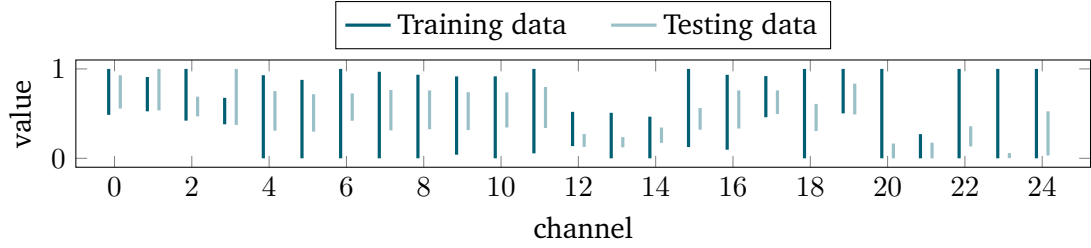
Figure A.2 shows a different entity of SMD that was excluded (machine-3-11). The displayed subsection in Figure A.2a shows an anomaly in channel 4 of the test data and the corresponding label, spanning 65 time steps. The long-term effect of the anomaly is clearly visible, as the signal is not recovering from the sudden drop. Figure A.2b shows the correlation matrix of the training data, which exhibits a different behavior from that of machine-1-1 or machine-1-6 displayed in the main body.

A.3 Dataset Results

The networks are trained on different input sizes, indicated by the subscript with the model names in the following tables. Tables A.4 and A.6 show the F1-scores of the different models with varying input window sizes for entity machine-1-1 and machine-1-6 from SMD, respectively. For each configuration, we calculated four thresholds with the introduced algorithms. For each threshold, three different calculations to get the F1-scores were applied. Tables A.5 and A.7 show the corresponding values for AUROC and AUPRC for our approaches. AnomalyTr. corresponds to the AnomalyTransformer architecture.



(a) Value range of entity machine-1-1 of the SMD dataset.



(b) Value range of the PSM dataset.

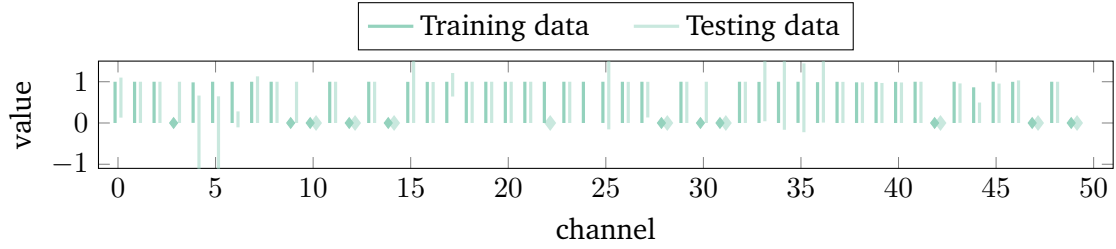
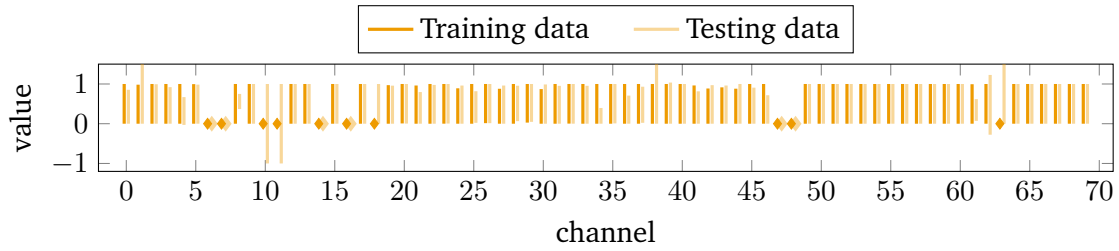
(c) Value range of the SWaT dataset. The values are displayed in the range $[-1, 1.5]$ for illustration purposes.(d) Value range of the WADI dataset. The values are displayed in the range $[-1.0, 1.5]$ for illustration purposes. For readability, the first 70 channels are presented here.

Figure A.1: Channel-wise value range of nominal data of different datasets. Diamonds represent constant values.

Table A.2: Entitites of the SMAP dataset.

Entity	Training	Testing	%	Entity	Training	Testing	%
A-1	2,880	8,640	0.98	E-6	2,880	8,300	0.80
A-2	2,648	7,914	1.40	E-7	2,769	8,310	3.38
A-3	2,736	8,205	2.27	E-8	2,880	8,532	7.30
A-4	2,690	8,080	1.37	E-9	2,880	8,302	4.23
A-5	705	4,693	1.09	E-10	2,880	8,505	3.79
A-6	682	4,453	0.92	E-11	2,880	8,514	3.46
A-7	2,879	8,631	27.82	E-12	2,880	8,512	6.85
A-8	762	8,375	45.44	E-13	2,880	8,640	3.06
A-9	762	8,434	45.83	F-1	2,869	8,584	1.18
B-1	2,435	8,044	0.88	F-2	2,861	8,626	34.28
D-1	2,849	8,509	38.30	F-3	2,880	8,376	0.49
D-2	2,880	8,595	49.08	G-1	2,820	8,469	1.43
D-3	2,880	8,640	37.92	G-2	2,478	7,361	0.56
D-4	2,833	8,473	38.33	G-3	2,624	7,907	0.64
D-5	2,561	7,628	0.67	G-4	2,551	7,632	0.41
D-6	2,594	7,884	1.03	G-6	2,881	8,640	1.17
D-7	2,583	7,642	35.36	G-7	2,446	8,029	3.34
D-8	2,602	7,874	0.65	P-1	2,872	8,505	8.83
D-9	2,583	7,406	15.61	P-2	2,821	8,209	13.66
D-11	2,611	7,431	0.82	P-3	2,855	8,493	15.73
D-12	312	7,918	34.60	P-4	2,609	7,783	5.69
D-13	1,490	7,663	2.10	P-7	2,853	8,071	20.46
E-1	2,880	8,516	5.97	R-1	2,874	7,244	1.12
E-2	2,880	8,532	16.39	S-1	2,818	7,331	6.11
E-3	2,880	8,307	38.68	T-1	2,875	8,612	17.84
E-4	2,880	8,354	33.66	T-2	2,855	8,625	20.70
E-5	2,880	8,294	3.87	T-3	2,876	8,579	2.14

Table A.3: Overview of the entities of SMD.

Entity	Training	Testing	%	Entity	Training	Testing	%
machine-1-1	28,479	28,479	9.46	machine-2-7	23,696	23,696	1.76
machine-1-2	23,694	23,694	2.29	machine-2-8	23,702	23,703	0.68
machine-1-3	23,702	23,703	3.45	machine-2-9	28,722	28,722	6.11
machine-1-4	23,706	23,707	3.04	machine-3-1	28,700	28,700	1.07
machine-1-5	23,705	23,706	0.42	machine-3-2	23,702	23,703	4.68
machine-1-6	23,688	23,689	15.65	machine-3-3	23,703	23,703	2.67
machine-1-7	23,697	23,697	10.12	machine-3-4	23,687	23,687	4.12
machine-1-8	23,698	23,699	3.22	machine-3-5	23,690	23,691	1.80
machine-2-1	23,693	23,694	4.94	machine-3-6	28,726	28,726	4.16
machine-2-2	23,699	23,700	11.95	machine-3-7	28,705	28,705	1.51
machine-2-3	23,688	23,689	1.14	machine-3-8	28,703	28,704	4.78
machine-2-4	23,689	23,689	7.15	machine-3-9	28,713	28,713	1.06
machine-2-5	23,688	23,689	4.14	machine-3-10	23,692	23,693	4.42
machine-2-6	28,743	28,743	1.48	machine-3-11	28,695	28,696	0.69

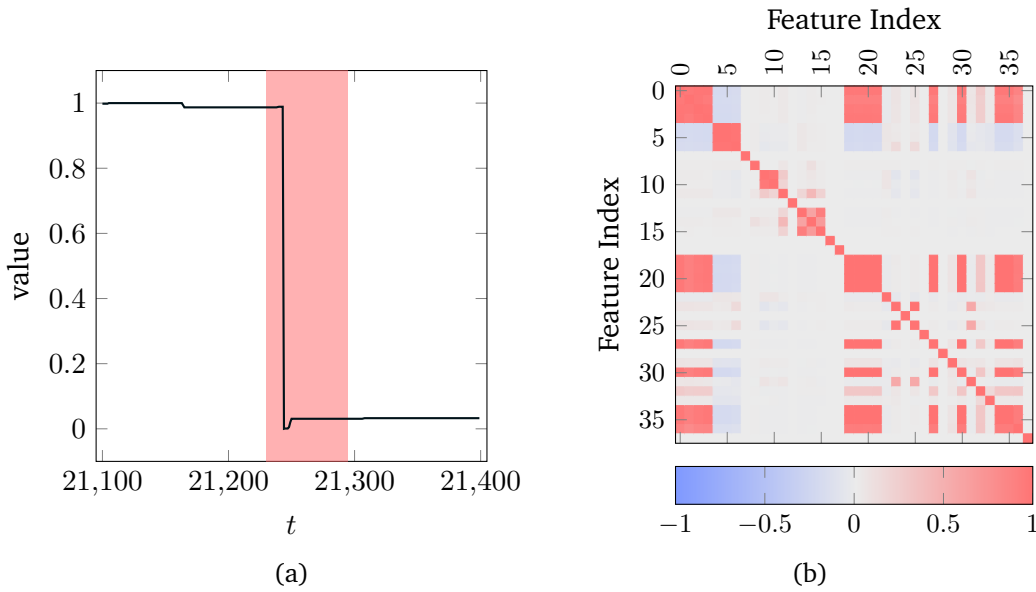


Figure A.2: Plots of machine-3-11 from SMD: (a) Segment of the fourth channel of test data. The highlighted area is the labeled anomaly. (b) Feature correlations of the training data.

Table A.4: Evaluation on SMD machine-1-1 with different methods of F1-score calculations and varying threshold algorithms. Best point-wise F1-scores for every non-iterative thresholding algorithm are highlighted in bold.

	POT			Gap			static			best		
	F1	F1 _{PA}	F1 _{PA%K}	F1	F1 _{PA}	F1 _{PA%K}	F1	F1 _{PA}	F1 _{PA%K}	F1	F1 _{PA}	F1 _{PA%K}
AE-GAN ₆₄	0.19	1.00	0.28	0.55	0.84	0.70	0.16	0.78	0.25	0.70	0.81	0.78
AE-GAN ₁₂₈	0.24	0.98	0.35	0.42	0.77	0.57	0.17	0.74	0.26	0.63	0.69	0.68
AE-GAN ₂₅₆	0.20	1.00	0.30	0.39	0.89	0.54	0.17	0.77	0.26	0.64	0.75	0.72
AE-GAN ₅₁₂	0.32	0.94	0.46	0.38	0.76	0.53	0.17	0.73	0.27	0.56	0.60	0.59
AE-GAN _{1,024}	0.30	0.95	0.44	0.38	0.76	0.53	0.17	0.72	0.27	0.45	0.59	0.54
Sparse-AE-GAN ₆₄	0.31	0.94	0.45	0.38	0.76	0.52	0.17	0.72	0.27	0.44	0.57	0.53
Sparse-AE-GAN ₁₂₈	0.30	0.94	0.43	0.35	0.78	0.49	0.18	0.87	0.28	0.35	0.82	0.50
Sparse-AE-GAN ₂₅₆	0.13	0.92	0.21	0.26	0.97	0.38	0.12	0.67	0.20	0.37	0.44	0.43
Sparse-AE-GAN ₅₁₂	0.33	0.93	0.47	0.37	0.77	0.51	0.18	0.74	0.27	0.43	0.56	0.52
Sparse-AE-GAN _{1,024}	0.29	0.96	0.43	0.38	0.78	0.52	0.17	0.73	0.27	0.44	0.58	0.53
Dual-AE-GAN ₆₄	0.23	0.99	0.35	0.58	0.89	0.74	0.18	0.80	0.28	0.75	0.87	0.84
Dual-AE-GAN ₁₂₈	0.17	1.00	0.26	0.56	0.88	0.71	0.16	0.78	0.25	0.73	0.85	0.82
Dual-AE-GAN ₂₅₆	0.16	1.00	0.25	0.72	0.87	0.83	0.17	0.79	0.26	0.74	0.86	0.83
Dual-AE-GAN ₅₁₂	0.14	1.00	0.22	0.54	0.82	0.69	0.20	0.80	0.30	0.54	0.81	0.69
Dual-AE-GAN _{1,024}	0.12	1.00	0.20	0.56	0.80	0.69	0.19	0.80	0.29	0.56	0.79	0.69
Swin1D-AD ₆₄	0.47	0.87	0.63	0.34	0.35	0.35	0.11	0.82	0.17	0.50	0.63	0.59
Swin1D-AD ₁₂₈	0.47	0.86	0.62	0.34	0.35	0.35	0.11	0.83	0.18	0.49	0.57	0.55
Swin1D-AD ₂₅₆	0.45	0.88	0.61	0.34	0.35	0.35	0.12	0.84	0.19	0.56	0.68	0.65
Swin1D-AD ₅₁₂	0.45	0.85	0.60	0.30	0.31	0.30	0.11	0.82	0.18	0.50	0.61	0.58
Swin1D-AD _{1,024}	0.64	0.78	0.73	0.29	0.29	0.29	0.11	0.81	0.18	0.64	0.83	0.75
EncoDiff1D ₆₄	0.19	0.98	0.28	0.30	0.90	0.44	0.10	0.86	0.17	0.37	0.81	0.51
EncoDiff1D ₁₂₈	0.23	0.99	0.35	0.31	0.85	0.44	0.17	0.87	0.26	0.48	0.57	0.55
EncoDiff1D ₂₅₆	0.21	0.97	0.32	0.37	0.93	0.52	0.20	0.74	0.31	0.52	0.82	0.66
EncoDiff1D ₅₁₂	0.19	1.00	0.29	0.46	0.89	0.62	0.15	0.85	0.23	0.52	0.66	0.60
EncoDiff1D _{1,024}	0.15	1.00	0.23	0.33	0.80	0.48	0.16	0.75	0.25	0.43	0.60	0.53
AnomalyTr. ₆₄	0.03	0.96	0.04	0.03	0.83	0.05	0.17	0.17	0.17	0.03	0.99	0.02
AnomalyTr. ₁₂₈	0.01	0.98	0.02	0.02	0.86	0.03	0.17	0.17	0.17	0.01	0.34	0.00
AnomalyTr. ₂₅₆	0.01	0.99	0.01	0.01	0.89	0.01	0.17	0.17	0.17	0.01	1.00	0.01
AnomalyTr. ₅₁₂	0.01	1.00	0.01	0.01	0.92	0.01	0.17	0.17	0.17	0.01	1.00	0.01
AnomalyTr. _{1,024}	0.01	0.91	0.01	0.03	0.97	0.04	0.18	0.18	0.18	0.01	0.91	0.01
TranAD ₆₄	0.03	0.84	0.05	0.56	0.59	0.59	0.17	0.17	0.17	0.04	1.00	0.07
TranAD ₁₂₈	0.03	0.84	0.05	0.56	0.60	0.60	0.17	0.17	0.17	0.04	1.00	0.07
TranAD ₂₅₆	0.02	0.84	0.05	0.61	0.68	0.66	0.17	0.17	0.17	0.04	1.00	0.07
ImDiffusion ₆₄	0.01	0.73	0.01	0.06	1.00	0.10	0.17	0.18	0.18	0.30	0.63	0.43
ImDiffusion ₁₂₈	0.01	0.73	0.01	0.07	1.00	0.12	0.17	0.18	0.18	0.27	0.87	0.40
ImDiffusion ₂₅₆	0.01	0.73	0.01	0.04	0.99	0.08	0.17	0.17	0.17	0.20	0.31	0.26

Table A.5: Evaluation of different algorithms on SMD machine-1-1 with varying threshold algorithms. AUROC and AUPRC scores are shown here. Best scores for every non-iterative thresholding algorithm are highlighted in bold.

	POT		Gap		static		best	
	AUROC	AUPRC	AUROC	AUPRC	AUROC	AUPRC	AUROC	AUPRC
AE-GAN ₆₄	0.55	0.18	0.74	0.35	0.54	0.12	0.87	0.51
AE-GAN ₁₂₈	0.57	0.19	0.68	0.23	0.54	0.11	0.89	0.45
AE-GAN ₂₅₆	0.55	0.19	0.64	0.23	0.54	0.12	0.86	0.45
AE-GAN ₅₁₂	0.60	0.21	0.66	0.21	0.54	0.12	0.89	0.38
AE-GAN _{1,024}	0.59	0.21	0.66	0.20	0.54	0.12	0.78	0.26
Sparse-AE-GAN ₆₄	0.60	0.21	0.66	0.20	0.54	0.11	0.77	0.25
Sparse-AE-GAN ₁₂₈	0.59	0.19	0.64	0.18	0.55	0.12	0.63	0.19
Sparse-AE-GAN ₂₅₆	0.54	0.16	0.57	0.19	0.52	0.10	0.77	0.22
Sparse-AE-GAN ₅₁₂	0.60	0.21	0.65	0.20	0.55	0.12	0.76	0.24
Sparse-AE-GAN _{1,024}	0.59	0.20	0.65	0.20	0.55	0.12	0.77	0.25
Dual-AE-GAN ₆₄	0.57	0.20	0.74	0.40	0.55	0.13	0.87	0.58
Dual-AE-GAN ₁₂₈	0.55	0.18	0.73	0.36	0.54	0.12	0.87	0.55
Dual-AE-GAN ₂₅₆	0.54	0.17	0.85	0.55	0.54	0.12	0.87	0.57
Dual-AE-GAN ₅₁₂	0.54	0.16	0.74	0.34	0.56	0.14	0.75	0.34
Dual-AE-GAN _{1,024}	0.53	0.16	0.77	0.35	0.55	0.13	0.77	0.35
Swin1D-AD ₆₄	0.68	0.28	0.79	0.20	0.53	0.11	0.80	0.31
Swin1D-AD ₁₂₈	0.68	0.28	0.79	0.20	0.53	0.11	0.83	0.30
Swin1D-AD ₂₅₆	0.67	0.27	0.79	0.20	0.53	0.12	0.83	0.36
Swin1D-AD ₅₁₂	0.68	0.26	0.74	0.17	0.53	0.11	0.81	0.31
Swin1D-AD _{1,024}	0.83	0.44	0.73	0.17	0.52	0.11	0.81	0.45
EncoDiff1D ₆₄	0.55	0.16	0.60	0.18	0.52	0.10	0.64	0.20
EncoDiff1D ₁₂₈	0.57	0.20	0.60	0.17	0.54	0.12	0.82	0.30
EncoDiff1D ₂₅₆	0.56	0.17	0.62	0.24	0.56	0.13	0.73	0.31
EncoDiff1D ₅₁₂	0.55	0.19	0.68	0.29	0.54	0.11	0.80	0.32
EncoDiff1D _{1,024}	0.54	0.17	0.62	0.18	0.54	0.12	0.75	0.24

Table A.6: Evaluation on SMD machine-1-6 with different methods of F1-score calculations and varying threshold algorithms. Best point-wise F1-scores for every non-iterative thresholding algorithm are highlighted bold.

	POT			Gap			static			best		
	F1	F1 _{PA}	F1 _{PA%K}	F1	F1 _{PA}	F1 _{PA%K}	F1	F1 _{PA}	F1 _{PA%K}	F1	F1 _{PA}	F1 _{PA%K}
AE-GAN ₆₄	0.15	0.92	0.22	0.75	0.99	0.88	0.27	0.97	0.38	0.89	0.99	0.93
AE-GAN ₁₂₈	0.65	0.92	0.80	0.32	0.32	0.32	0.26	0.95	0.38	0.89	0.98	0.92
AE-GAN ₂₅₆	0.35	0.92	0.49	0.50	0.53	0.52	0.24	0.94	0.35	0.89	0.99	0.93
AE-GAN ₅₁₂	0.88	0.99	0.92	0.31	0.31	0.31	0.24	0.95	0.36	0.89	0.99	0.93
AE-GAN _{1,024}	0.66	0.72	0.67	0.30	0.32	0.31	0.22	0.94	0.32	0.88	0.99	0.92
Sparse-AE-GAN ₆₄	0.16	0.92	0.24	0.54	0.58	0.54	0.27	0.97	0.38	0.90	0.99	0.93
Sparse-AE-GAN ₁₂₈	0.49	0.92	0.64	0.45	0.49	0.45	0.26	0.93	0.38	0.75	0.88	0.81
Sparse-AE-GAN ₂₅₆	0.47	0.92	0.63	0.43	0.48	0.44	0.27	0.94	0.39	0.78	0.91	0.84
Sparse-AE-GAN ₅₁₂	0.60	0.88	0.75	0.52	0.58	0.53	0.26	0.93	0.37	0.64	0.81	0.72
Sparse-AE-GAN _{1,024}	0.55	0.92	0.71	0.41	0.45	0.41	0.27	0.94	0.39	0.81	0.93	0.86
Dual-AE-GAN ₆₄	0.23	0.92	0.34	0.38	0.41	0.38	0.27	0.98	0.93	0.89	0.99	0.93
Dual-AE-GAN ₁₂₈	0.86	0.97	0.90	0.31	0.31	0.31	0.22	0.94	0.31	0.86	0.96	0.90
Dual-AE-GAN ₂₅₆	0.82	0.99	0.91	0.31	0.31	0.31	0.25	0.97	0.36	0.88	0.99	0.92
Dual-AE-GAN ₅₁₂	0.89	1.00	0.93	0.31	0.31	0.31	0.27	0.97	0.39	0.90	1.00	0.93
Dual-AE-GAN _{1,024}	0.86	0.99	0.92	0.31	0.31	0.31	0.26	0.95	0.37	0.89	0.98	0.92
Swin1D-AD ₆₄	0.85	0.98	0.91	0.31	0.31	0.31	0.25	0.91	0.36	0.88	0.98	0.92
Swin1D-AD ₁₂₈	0.79	0.98	0.89	0.31	0.31	0.31	0.26	0.92	0.38	0.89	0.99	0.93
Swin1D-AD ₂₅₆	0.88	0.98	0.92	0.31	0.31	0.31	0.25	0.92	0.36	0.88	0.98	0.92
Swin1D-AD ₅₁₂	0.88	0.98	0.92	0.31	0.31	0.31	0.24	0.91	0.35	0.88	0.98	0.92
Swin1D-AD _{1,024}	0.88	0.98	0.92	0.31	0.31	0.31	0.22	0.91	0.32	0.88	0.98	0.92
EncoDiff1D ₆₄	0.01	0.92	0.01	0.32	0.93	0.46	0.17	0.89	0.26	0.66	0.83	0.75
EncoDiff1D ₁₂₈	0.02	0.92	0.03	0.65	0.98	0.80	0.18	0.90	0.27	0.77	0.89	0.83
EncoDiff1D ₂₅₆	0.06	0.92	0.09	0.77	0.90	0.83	0.20	0.92	0.30	0.77	0.90	0.83
EncoDiff1D ₅₁₂	0.05	0.92	0.09	0.78	0.87	0.81	0.21	0.95	0.31	0.85	0.97	0.91
EncoDiff1D _{1,024}	0.02	0.92	0.03	0.86	0.98	0.92	0.21	0.95	0.30	0.86	0.97	0.91
AnomalyTr. ₆₄	0.03	0.97	0.05	0.03	0.97	0.05	0.27	0.27	0.27	0.04	0.95	0.06
AnomalyTr. ₁₂₈	0.02	0.96	0.02	0.02	0.92	0.04	0.27	0.27	0.27	0.02	0.97	0.02
AnomalyTr. ₂₅₆	0.00	0.00	0.00	0.02	0.92	0.03	0.27	0.27	0.27	0.01	0.98	0.02
AnomalyTr. ₅₁₂	0.00	0.92	0.01	0.03	0.93	0.04	0.27	0.27	0.27	0.01	0.99	0.03
AnomalyTr. _{1,024}	0.00	0.92	0.01	0.04	0.98	0.06	0.27	0.27	0.27	0.01	0.99	0.02
TranAD ₆₄	0.18	0.92	0.27	0.31	0.31	0.31	0.27	0.27	0.27	0.85	1.00	0.92
TranAD ₁₂₈	0.25	0.92	0.36	0.31	0.31	0.31	0.27	0.27	0.27	0.88	1.00	0.93
TranAD ₂₅₆	0.23	0.92	0.33	0.31	0.31	0.31	0.27	0.27	0.27	0.88	1.00	0.93
ImDiffusion ₆₄	0.01	0.22	0.02	0.05	1.00	0.07	0.27	0.28	0.28	0.59	0.85	0.73
ImDiffusion ₁₂₈	0.01	0.22	0.02	0.04	0.99	0.06	0.27	0.28	0.28	0.65	0.86	0.77
ImDiffusion ₂₅₆	0.01	0.21	0.02	0.04	0.99	0.06	0.27	0.27	0.27	0.46	0.60	0.54

Table A.7: Evaluation of different algorithms on SMD machine-1-6 with varying threshold algorithms. AUROC and AUPRC scores are shown here. Best scores for every non-iterative thresholding algorithm are highlighted in bold.

	POT		Gap		static		best	
	AUROC	AUPRC	AUROC	AUPRC	AUROC	AUPRC	AUROC	AUPRC
AE-GAN ₆₄	0.54	0.23	0.80	0.66	0.58	0.25	0.91	0.84
AE-GAN ₁₂₈	0.74	0.56	0.60	0.19	0.57	0.24	0.90	0.82
AE-GAN ₂₅₆	0.60	0.33	0.79	0.33	0.57	0.22	0.90	0.82
AE-GAN ₅₁₂	0.89	0.82	0.58	0.18	0.57	0.23	0.91	0.83
AE-GAN _{1,024}	0.85	0.48	0.57	0.18	0.56	0.21	0.90	0.82
Sparse-AE-GAN ₆₄	0.54	0.23	0.81	0.36	0.58	0.25	0.91	0.84
Sparse-AE-GAN ₁₂₈	0.66	0.43	0.75	0.28	0.57	0.23	0.86	0.60
Sparse-AE-GAN ₂₅₆	0.65	0.41	0.74	0.27	0.57	0.23	0.87	0.65
Sparse-AE-GAN ₅₁₂	0.72	0.47	0.80	0.35	0.57	0.22	0.80	0.46
Sparse-AE-GAN _{1,024}	0.69	0.47	0.71	0.25	0.58	0.23	0.87	0.69
Dual-AE-GAN ₆₄	0.55	0.24	0.65	0.21	0.57	0.25	0.91	0.84
Dual-AE-GAN ₁₂₈	0.89	0.77	0.58	0.18	0.56	0.21	0.90	0.77
Dual-AE-GAN ₂₅₆	0.85	0.74	0.58	0.18	0.57	0.24	0.90	0.82
Dual-AE-GAN ₅₁₂	0.61	0.20	0.58	0.18	0.58	0.25	0.91	0.84
Dual-AE-GAN _{1,024}	0.64	0.21	0.58	0.18	0.57	0.23	0.91	0.82
Swin1D-AD ₆₄	0.87	0.78	0.58	0.18	0.57	0.21	0.90	0.82
Swin1D-AD ₁₂₈	0.83	0.71	0.58	0.18	0.57	0.22	0.90	0.83
Swin1D-AD ₂₅₆	0.89	0.82	0.58	0.18	0.57	0.22	0.90	0.82
Swin1D-AD ₅₁₂	0.89	0.82	0.58	0.18	0.56	0.21	0.90	0.82
Swin1D-AD _{1,024}	0.89	0.82	0.58	0.18	0.56	0.20	0.90	0.82
EncoDiff1D ₆₄	0.50	0.16	0.60	0.31	0.54	0.18	0.81	0.49
EncoDiff1D ₁₂₈	0.51	0.17	0.75	0.54	0.54	0.18	0.87	0.63
EncoDiff1D ₂₅₆	0.52	0.18	0.87	0.63	0.55	0.19	0.86	0.64
EncoDiff1D ₅₁₂	0.51	0.18	0.89	0.64	0.56	0.21	0.89	0.76
EncoDiff1D _{1,024}	0.50	0.16	0.89	0.78	0.55	0.21	0.90	0.78



INDUSTRIAL DATASET DETAILS

B.1 KIWA

Table B.1: Channel definitions of the KIWA dataset.

subgroup	variable	description
cap, needle, X, Y	AW_ActPos	actual position
	AW_SetPos	set position
	AW_CntSta	control status
	AW_Ao_Out	analogue output
	AW_SpdMod	speed mode
acc1, acc2	X	acceleration in X
	Y	acceleration in Y
	Z	acceleration in Z

B.2 DigiSeal

The different channels of the DigiSeal dataset are detailed here. The feature numbers in Table B.2 and B.3 correspond to the ones used in the main body of this work. Multiple feature numbers correspond to DE, NDE, Inlet, and Outlet, respectively.

Table B.2: Channel definitions of sensor_setup1 of the DigiSeal dataset.

subgroup1	subgroup2	feature number	description/measuring point
DE/NDE	temperature	0/7	bearing y
		1/8	bearing z
		2/9	sealface
	acceleration	3/10	bearing
		4/11	x
		5/12	y
		6/13	z
		14	axial displacement
		15	speed

Table B.3: Channel definitions of sensor_setup2 of the DigiSeal dataset.

subgroup1	subgroup2	feature number	description/measuring point
DE/NDE	Inlet/Outlet	16/21/26/31	temperature of the fluid
		17/22/27/32	pressure of the fluid
		18/23/28/33	acceleration in x
		19/24/29/34	acceleration in y
		20/25/30/35	acceleration in z