CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# Representation Learning for Clustering and Analysis of High-Dimensional Time-Series Data

Master's thesis in Computer Science - Algorithms, Languages and Logic

JOEL LEIDITZ THORSSON,
JONATHAN NAUMANEN

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

# Representation Learning for Clustering and Analysis of High-Dimensional Time-Series Data

JOEL LEIDITZ THORSSON,
JONATHAN NAUMANEN

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2026

Representation Learning for Clustering and Analysis of High-Dimensional Time-Series Data
JOEL LEIDITZ THORSSON, JONATHAN NAUMANEN



Supervisor: Martin Hilgendorf & Quang Vinh Ngo,
Department of Computer Science and Engineering
Ivor Curik, Ericsson

Examiner: Marina Papatriantafilou,
Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Representation Learning for Clustering and Analysis of High-Dimensional Time-Series Data

JOEL LEIDITZ THORSSON, JONATHAN NAUMANEN
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

# Abstract

Modern software systems generate large volumes of high-dimensional multivariate time-series data during testing and operation. In industrial settings such as continuous integration and continuous deployment (CI/CD) pipelines, manually diagnosing recurring failures across thousands of performance metrics is time-consuming and error-prone. This thesis investigates whether representation learning can produce stable, interpretable, and diagnostically useful groupings of anomalous behaviour in such data. This allows for recurring behavioural patterns to be identified and related to meaningful system characteristics, thereby supporting failure investigation at scale. We develop and compare several autoencoder architectures, including standard, variational, Long Short-Term Memory-based (LSTM), and Transformer-based variants, against traditional dimensionality reduction baselines and state-of-the-art anomaly detection models. Each architecture encodes multivariate time windows into a compact latent representation, from which reconstruction error is derived as an anomaly score and clustering is subsequently performed. We evaluate these methods across four dimensions: anomaly detection performance, clustering stability under perturbation, interpretability of the resulting cluster structure, and computational cost. Experiments are conducted on three established benchmark datasets: Server Machine Dataset (SMD), Secure Water Treatment System (SWaT), and Soil Moisture Active Passive (SMAP) and an unlabelled industrial dataset from an Ericsson CI/CD testing pipeline. No single model consistently outperforms across all datasets. LSTM-based and probabilistic architectures achieve the highest anomaly detection scores, yet traditional methods retain approximately 90% of peak performance at a fraction of the computational cost. Clustering stability is high on the benchmark datasets, with simpler models frequently matching more complex architectures under perturbation. This means that the models capture meaningful structure instead of noise which is important for downstream analysis. The LSTM encoder combined with Uniform Manifold Approximation and Projection (UMAP) with K-Means clustering produces the most interpretable groupings, with cluster prototypes showing clearly distinct feature-level patterns. In the industrial case study, the approach groups test runs by performance-metric dynamics rather than textual error similarity, offering a complementary perspective for fault diagnosis. Across all tasks, the relationship between model complexity and performance proves non-linear: moving beyond linear projections yields meaningful gains, but further architectural complexity delivers diminishing returns.

Keywords: AI, Autoencoder, Time-Series, Multivariate, Latent Space, Clustering, Representation Learning, Anomaly Detection, Machine Learning

# Acknowledgements

We want to express our gratitude to our supervisors, Quang Vinh Ngo and Martin Hilgendorf, and our examiner Marina Papatriantafilou, for their guidance throughout the project. We would also like to thank our research group for the insightful feedback we have had the privilege to take part in during the course of this thesis.

A big thanks to the Data-Driven Solutions team at Ericsson for allowing us to do our thesis with them and giving us insight into working with data in real-world applications. Especially to our industrial supervisor, Ivor Curik, for the engaging discussions, inspiring work ethic, and words of wisdom.

Finally, we would like to extend our thanks to our family and loved ones for their continued encouragement and support.

Joel Leiditz Thorsson and Jonathan Naumanen, Gothenburg, June 2026

# Notation table

| Symbol | Description |
|---|---|
| $X = \{x_1, \ldots, x_n\}$ | Dataset of time windows |
| $x_i \in \mathbb{R}^{T \times d}$ | $i$-th multivariate time window |
| $x \in \mathbb{R}^{T \times d}$ | Generic time window |
| $\hat{x}_i$ | Reconstruction of $x_i$ |
| $T$ | Length of each time window |
| $d$ | Number of features (dimensions) |
| $n$ | Number of windows in the dataset |
| $x_{ij} \in \mathbb{R}^T$ | Time-series of feature $j$ in window $x_i$ |
| $z$ | Latent representation of a window |
| $\mathcal{Z}$ | Set of latent representations $\{z_1, \ldots, z_n\}$ |
| $f_\theta(\cdot)$ | Encoder function |
| $g_\theta(\cdot)$ | Decoder function |
| $R(x)$ | Reconstruction error function |
| $\tau$ | Anomaly detection threshold |
| $K$ | Number of clusters |
| $c(\cdot)$ | Cluster assignment function, $c : \mathcal{Z} \to \{1, \ldots, K\}$ |
| $\mathcal{C}_k$ | Set of instances assigned to cluster $k$ |

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

Modern data-intensive systems can generate large amounts of data during runtime. Effective monitoring and analysis tools are essential to understand system behaviour, assess performance and detect faults early in the development and deployment process. Prior research has shown that performance metrics and hardware counters can contain predictive signals of system failures and anomalous behaviour [5, 6]. These metrics include CPU utilisation, memory usage, cache misses, and I/O activity that reflect the internal behaviour of a system during execution. Specifically, deviations in these measurements can be early signs of system failures or performance degradation, which makes them useful signals for identifying anomalous system behaviour at an early stage [5, 7]. However, analysing large-scale data remains challenging due to the volume and dimensionality, which makes storing and exact computation very resource-intensive, and in some cases, infeasible.

In industrial software development, performance testing is commonly used to evaluate system performance and behaviour under different workloads before deploying new code into production. These tests typically measure system metrics such as response time and resource utilisation to detect performance regressions, system instability, or abnormal behaviour introduced by new changes [5]. When these metrics exhibit anomalous behaviour, troubleshooting often involves manual inspection and cross-referencing of logs and performance metrics to determine the cause of the problem. This process is time-consuming and relies on the developer's domain expertise, particularly when tasked to recognise similarities in anomalous behaviour across multiple test-runs [6, 7, 8].

Recent advances in the field of representation learning for time-series data show that high-dimensional temporal behaviour can be captured in a latent space while preserving meaningful structure [9, 10]. Such learned representations have successfully been used for anomaly detection and system monitoring tasks using autoencoder-based architectures, a type of neural network that learns to encode data into a latent representation and reconstruct it back to its original form [1, 11, 12]. In related fields, dimensionality reduction techniques have been used to project high-dimensional data into lower-dimensional representations, after which clustering techniques have been used to identify structural patterns in large-scale time-series data [7, 13, 14].

This thesis investigates whether learned representations can be used to produce stable and interpretable groupings of structural patterns in high-dimensional time-series data. To address this, we utilise different variants of autoencoders.

This work bridges two traditionally separate fields: Anomaly Detection (AD) and Time-Series clustering. While reconstruction error in autoencoders is a well-established metric for identifying anomalies, we also explore the structural utility of the learned latent space.

Recent large-scale evaluations have shown that complex deep learning models are not always superior to simpler traditional methods and are often remarkably sensitive to their parameter settings [15]. Thus, our work provides a critical evaluation of both traditional linear projections and non-linear deep learning architectures to determine if increased complexity actually translates into more stable and useful groupings of data.

We extend the study by looking at industrial-scale data in a case study. There, the goal is to determine if the latent representations allow failed test runs to be grouped according to the underlying system behaviours that produced them, and group similar failures together.

The report begins by presenting the background and theoretical foundations for the proposed approach. It then introduces the industrial problem context, motivates the work, and formulates the research questions. This is followed by the methodology, which describes the formal problem formulation, the proposed approach, related challenges, and the model architectures considered. The evaluation section outlines the experimental methodology and setup before presenting and discussing the results with respect to the research questions. The report then reviews related work to position the proposed approach within the existing literature. Finally, the thesis concludes by summarising the main findings, discussing their industrial applicability, and outlining directions for future research.

# 2

# Background

This chapter provides background and description of different families of algorithms that will be utilised and evaluated in the methodology.

## 2.1 Dimensionality Reduction Techniques

Storing and grouping common causes of failures allows developers to efficiently find and diagnose recurrent issues and reduce the time spent on manual analysis. However, identifying when two errors share the same underlying feature characteristics is non-trivial when dealing with high-dimensional time-series data, as storing and comparing raw data is impractical and does not scale well as the number of features and test-runs increases. A common approach to summarising features is to aggregate them over time windows, usually computing statistics such as the mean, variance and quantiles [5, 16, 17]. While these methods are computationally efficient and easy to interpret, they may fail to capture interactions between features that are important to distinguishing between failure types. To address these issues, dimensionality reduction techniques such as Principal Component Analysis (PCA), feature hashing and random projections [18, 19] have been studied for their ability to project high-dimensional data to lower-dimensional spaces while preserving certain properties of the original data.

### 2.1.1 Principal Component Analysis

PCA works by identifying the directions of maximum variance in the data, forming a new coordinate system defined by the principal components, and then reducing the dimensionality by projecting the data onto only the first few components [20, 21]. An example of how PCA forms a new coordinate system defined by its principal components can be seen in Figure 2.1.

### 2.1.2 Random Projections

Random projections is another approach to dimensionality reduction where high-dimensional data is projected onto a lower-dimensional subspace by multiplying the original data by a randomly generated projection matrix. Based on the JohnsonLindenstrauss lemma, random projections approximately preserve pairwise distances

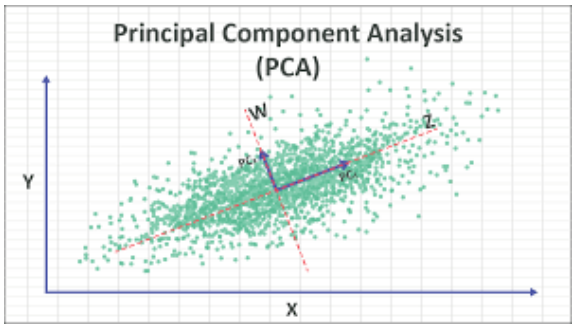


**Figure 2.1:** PCA identifying the directions of maximum variance in the data (PC1 and PC2), forming a new coordinate system aligned with the data distribution.

between data points with high probability [19, 22]. Due to these properties, random projections are computationally efficient for large-scale data processing while still retaining important geometric relationships in the data [23].

While methods such as PCA and random projections are effective for reducing dimensionality while preserving important statistical properties of the data, they are limited to linear transformations of the input space [24]. In high-dimensional data, relations between features can be non-linear and therefore hard or impossible to catch using purely statistical aggregation or linear projections. This motivates the use of methods that can capture non-linear relations and retain that information in low-dimensional representations.

### 2.1.3 Kernel Principal Component Analysis

One variant that solves this is Kernel Principal Component Analysis (KPCA) that extends PCA by first projecting the original data to a higher-dimensional feature space using a kernel function before then performing PCA in that transformed space [25]. Unlike traditional PCA, KPCA is able to capture non-linear structure from the original data in the low dimensional representation. It achieves this through the *kernel trick*, where pairwise similarities between samples are computed using a kernel matrix without explicitly constructing the higher-dimensional feature space. In this transformed space, non-linear features from the original space *may* become linearly separable, which then allows PCA to extract components that preserves non-linear structure from the original data. The *kernel trick* allows for efficient modelling of complex relations without directly operating over the potentially very high-dimensional space,

An illustration of the difference between PCA and KPCA is shown in Figure 2.2. While PCA identifies linear directions of maximum variance in the original feature space, KPCA can uncover non-linear structures by implicitly transforming the data before projection.

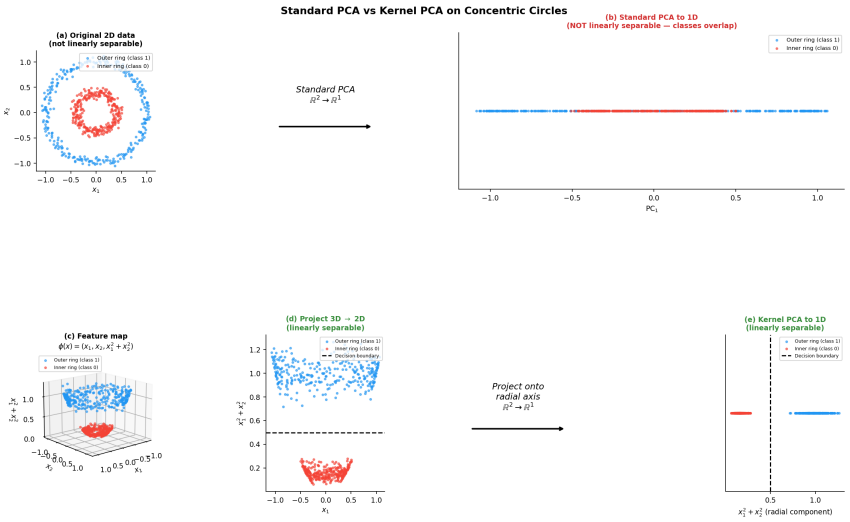


**Figure 2.2:** Comparison between PCA and KPCA. PCA identifies linear directions of variance in the original feature space, while KPCA captures non-linear structures by projecting the data into a higher-dimensional feature space using a kernel function before dimensionality reduction.

### 2.1.4 T-distributed Stochastic Neighbour Embedding

T-distributed Stochastic Neighbour Embedding (t-SNE) is a non-linear dimensionality reduction technique primarily used for visualisation of high-dimensional data in a low-dimensional space [26]. The method aims to preserve local neighbourhood structures by mapping similar high-dimensional data points to nearby points in a two- or three-dimensional embedding space, while dissimilar points are placed farther apart with high probability.

It works by modelling pairwise similarities between samples as probability distributions in both the high-dimensional and low-dimensional spaces and minimises the divergence between these distributions during optimisation [26]. This enables the method to reveal hidden structures, clusters, and relationships within complex datasets that may otherwise be difficult to interpret due to the high dimensions.

Although t-SNE is highly effective for visualising local structures and cluster formation, it is less effective at preserving global relationships between clusters. Consequently, distances between separated groups in the embedding space should be interpreted with caution.

### 2.1.5 Uniform Manifold Approximation and Projection

Uniform Manifold Approximation and Projection (UMAP) is a non-linear dimensionality reduction technique designed for both visualisation and manifold learning [27]. Compared to t-SNE, UMAP generally preserves more of the global structure of the data while also offering improved computational efficiency for large datasets.

Manifold learning is based on the assumption that high-dimensional data often lies on a lower-dimensional manifold embedded within the higher-dimensional feature space. The objective is therefore to identify and preserve the intrinsic geometric structure of the data while reducing dimensionality.

UMAP constructs a graph-based representation of the high-dimensional data manifold by modelling local relationships between neighbouring samples. It then optimises a low-dimensional embedding that preserves both local neighbourhood structures and aspects of the global topology of the dataset. This often results in embeddings that maintain meaningful cluster relationships while remaining computationally efficient.

Due to its ability to preserve both local and partial global structures, UMAP has become a popular choice for visualisation, preprocessing, and clustering of high-dimensional data [28, 29].

### 2.1.6 Complexity Analysis

The computational cost of dimensionality reduction techniques varies considerably depending on how they model the structure of data. This section compares the complexity of the different dimensionality reduction techniques.

#### 2.1.6.1 Complexity - PCA

PCA reduces dimensionality by computing the covariance matrix and performing an eigendecomposition. When the number of features $d$ is smaller than the number of samples $n$, PCA has a time complexity of $O(nd+d^2)$ [30]. Although efficient for many applications, the computational cost increases rapidly as the number of features grows since PCA explicitly models relationships between all feature dimensions.

#### 2.1.6.2 Complexity - KPCA

The extension that KPCA brings to PCA allows it to capture non-linear relationships. However, this flexibility comes at a significantly higher computational cost. For a dataset consisting of $n$ samples, KPCA constructs an $n \times n$ kernel matrix, where each entry represents the similarity between a pair of samples. Computing this matrix requires evaluating the kernel function for every pair of samples, resulting in $O(n^2)$ time and memory complexity. Furthermore, eigendecomposition of the kernel matrix typically requires $O(n^3)$ time [25, 30]. Consequently, KPCA scales with the number of samples rather than the number of features and can become computationally infeasible for large datasets.

#### 2.1.6.3 Complexity - Random Projections

Random Projections provides a considerably more scalable approach. Given a data matrix $X_{n\times d}$ and a random projection matrix $R_{d\times k}$, where $k \ll d$, dimensionality reduction is obtained through a matrix multiplication. This operation requires

$O(ndk)$ time and $O(nd + dk + nk)$ memory [22]. Since complexity grows approximately linearly with problem dimensions, random projections are generally the most computationally and memory-efficient of the three approaches.

#### 2.1.6.4 Complexity - t-SNE

Unlike the three methods mentioned above, which are commonly used for general-purpose dimensionality reduction, t-SNE is primarily designed for low-dimensional embedding and visualisation. Computing pairwise similarities between all $n$ data points in the high-dimensional space requires $O(n^2 d)$ time and $O(n^2)$ memory. The optimisation procedure, which typically relies on gradient descent, further increases the computational cost and generally scales as $O(n^2)$ per iteration [26].

To improve scalability, modern implementations of t-SNE often use approximations such as Barnes-Hut t-SNE or FFT-based methods, reducing the effective complexity to approximately $O(n \log n)$ per iteration for moderate embedding dimensions [31, 32]. However, despite these optimisations, t-SNE remains computationally demanding for large datasets and is therefore primarily used for visualisation rather than as a general-purpose dimensionality reduction method.

#### 2.1.6.5 Complexity - UMAP

UMAP addresses several of the scalability limitations of t-SNE while preserving both local and global structure in the data. It first constructs a $k$-nearest neighbour graph, which can be computed in approximately $O(n \log n)$ time using approximate nearest neighbour methods. The subsequent optimisation step, which aligns the low-dimensional embedding with the high-dimensional graph structure, scales approximately linearly with the number of data points, which is $O(n)$ per epoch. To store the nearest-neighbour graph requires $O(nk)$ space, which is considerably more space efficient than t-SNE [26, 27].

#### 2.1.6.6 Complexity - Trade-offs

Each method represents a different trade-off between computational efficiency and representational power. Random Projections offer linear scaling but are also limited to linear transformations, as discussed in Section 2.1.2. PCA captures the principal linear transformations but struggles with highly non-linear relationships, although at a substantial computational and memory cost. While t-SNE and UMAP are capable of preserving complex non-linear structures, they are primarily intended for embedding and visualisation rather than general-purpose feature extraction. This motivates the use of other, more flexible techniques that are capable of capturing non-linear patterns in high-dimensional, temporally dependent data. *Representation Learning* approaches, particularly neural network architectures such as *Autoencoders*, have been studied for this purpose and emerged as possible solutions.

## 2.2 Representation Learning and Autoencoders

The goal of representation learning is to train a model to extract information from high-dimensional data and transform it into a more compact representation. This representation can be expressed as a vector in a lower-dimensional latent space, usually referred to as an *embedding*, which aims to preserve structures present in the original data important for reconstruction and analysis tasks [33]. For multivariate time-series data, these structures may include temporal patterns, correlations between variables, and characteristic system behaviours that distinguish normal operation from anomalous events.

### 2.2.1 Autoencoders

Autoencoders are neural network architectures designed to learn compact representations by training the network to encode input data into a lower-dimensional latent space and then reconstruct, or decode, the original input from this representation [1, 24, 34, 35].

#### 2.2.1.1 Architecture

The traditional autoencoder architecture can be seen in Figure 2.3 and consists of two main components:

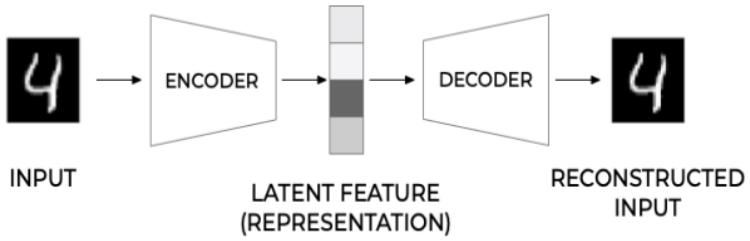


**Figure 2.3:** Figure of a basic autoencoder architecture [1].

**The *Encoder*** performs the dimensionality reduction step of the autoencoder by mapping the input vector to a lower-dimensional latent representation. Formally, let $x \in \mathbb{R}^d$ denote the input vector, let $k$ be the lower dimensionality where the vector is projected, such that $k \ll d$, and let $\theta$ denote the learnable parameters in the encoder's neural network. Then, the encoder function can be defined as:

$$f_\theta : \mathbb{R}^d \to \mathbb{R}^k$$

If we denote the latent representation of $x$ by $z \in \mathbb{R}^k$, the encoding step can be defined by:

$$z = f_\theta(x)$$

The latent representation $z$ is expected to capture the most informative structures of the data, such as correlations between variables and recurring temporal patterns, while discarding redundant or noisy information.

**The *Decoder*** takes the latent representation and attempts to reconstruct the original input. Formally, we can define the decoder function by:

$$g_\phi : \mathbb{R}^k \to \mathbb{R}^d$$

Where $\phi$ denotes the learnable parameters in the neural network of the decoder and $k$ and $d$ are the same dimensions as before. The output of the decoder can then be denoted by $\hat{x} \in \mathbb{R}^d$ and its relation to the decoder function and the latent representation $z$ can be described by:

$$\hat{x} = g_\phi(z)$$

The goal of training is therefore to learn a latent representation in dimension $k$ using the encoder and later reconstruct the original input through the decoder. These two are usually symmetrical in their structure and trained jointly [1].

#### 2.2.1.2 Training

The parameters corresponding to the encoder and decoder parts of the model, denoted $\theta$ and $\phi$ respectively, are learned during training by minimising the reconstruction loss between the input $x$ and the reconstructed output $\hat{x}$. Formally, we can define the training function by:

$$\min_{\theta,\phi} \ \mathcal{L}(x, \hat{x}) = \min_{\theta,\phi} \ \|x - g_\phi(f_\theta(x))\|$$

where $\mathcal{L}$ denotes the reconstruction loss. A well-trained autoencoder learns a compact representation of normal system behaviour, causing inputs that deviate from these patterns to produce larger reconstruction errors. To optimise the $\theta$ and $\phi$ parameters of the network, gradient-based methods such as stochastic gradient descent and its variants are often used for training [1].

To learn complex, non-linear relationships within high-dimensional data, autoencoders employ non-linear activation functions within their neural network architectures. These activation functions allow the encoder and decoder to approximate non-linear mappings between the input space and the latent representation.

The *Rectified Linear Unit (ReLU)* is the most commonly used activation function in modern architectures [36]. The ReLU function is defined by:

$$f(x) = \max(0, x)$$

By introducing non-linear activations between layers, the autoencoder can capture complex structures in the data that cannot be represented using purely linear transformations.

While the autoencoder learns deterministic mappings from input to representation, it does not impose any structure on the latent space. This leads to a resulting embedding that is only constrained by the reconstruction accuracy, which can lead to embeddings that form irregular or discontinuous regions in the latent space, limiting their usefulness for tasks that are dependent on the structure of the latent space such as clustering.

### 2.2.2 Variational Autoencoders

Variational Autoencoders (VAEs) extend the standard autoencoder by introducing an additional loss term penalizing the learned probability distribution of the latent space, encouraging it to conform to a standard normal distribution. This should in turn lead to a smooth, continuous latent space. Instead of mapping an input $x$ deterministically to a latent vector $z$ as in the normal autoencoder, the encoders learns the parameters of a probability distribution over the latent space [37].

The encoder of the VAE produces two outputs per input $x$:

- a mean vector($\mu(x)$)
- and a variance vector $\sigma^2(x)$

together, these define the approximate posterior distribution:

$$z \sim \mathcal{N}(\mu(x), \sigma^2(x))$$

The VAE is then trained by optimising a combined loss function consisting of the regular reconstruction term as well as the regularisation term:

$$\min_{\theta,\phi} \mathcal{L}(x, \hat{x}) = \min_{\theta,\phi} \left(\|x - g_\phi(f_\theta(x))\|^2 + \beta D_{KL}(q_\theta(z|x)\|p(z))\right)^1$$

where:

- the first term, $\|x - g_\phi(f_\theta(x))\|^2$, is the reconstruction loss
- the second term $\beta D_{KL}(q_\theta(z|x)\|p(z))$ regularises the latent distribution (KL-loss)
- $q_\theta(z|x)$ is the encoders approximate posterior
- $p(z)$ is typically a standard normal prior

### 2.2.3 Temporal Autoencoder Variants

While standard autoencoders often operate over static input vectors, many real world scenarios work on data where sequential observations are temporally dependent, meaning that the state of the system at a given time is influenced by previous observations. One common approach to capturing this time-dependency is by extending the autoencoder's architecture with sequence modelling components.

#### 2.2.3.1 Recurrent Autoencoders

Recurrent neural networks (RNNs) are able to process sequential data by having memory of past inputs to maintain an internal *hidden state* that acts as a summary of information from previous time steps that it updates over time. The hidden state can be interpreted as a learned representation of the sequence history. At each time step $t$, the network receives an input $X_t$ and updates its hidden state $H_t$ based on both the current input and the previous hidden state $H_{t-1}$. Thanks to this property,

[1] In the VAE setting, the encoder $f_\theta(x)$ parameterises a probability distribution over the latent variables rather than directly producing a deterministic latent vector, as in the standard autoencoder formulation.

the RNN can incorporate information from past observations when processing new inputs.

One prominent variant is the **Long-Short-Term-Memory (LSTM)** architecture, which can maintain both long-term and short-term memory thanks to a set of gating mechanisms that regulate the flow of information over time. Cells are the fundamental building blocks of RNNs that maintain the hidden states. An LSTM cell differs from a simple RNN cell in that it maintains two states: the hidden state $H_t$ and a *cell state* $C_t$. The hidden state plays the same role as in standard RNNs as the short-term representation of the sequence history that can be passed to the next layer. The cell-state acts as the long-term memory of the LSTM, it carries information across multiple time steps and enables the LSTM model to retain relevant information over long time horizons while still updating its short-term representation (hidden state) with new inputs.

An LSTM network cell typically contains three gates, a *forget gate*, an *input gate*, and an *output gate*. Combined, these gates allow the model to selectively choose which information to retain, update or discard from previous time steps when processing the current input. The forget gate regulates which information is discarded from the previous cell state, while the input gate determines how new information should be incorporated into the resulting memory cell. The output gate controls how much of the internal cell state contributes to the hidden representation passed to the next time step. An example of the internal structure of an LSTM cell can be seen in Figure 2.4.

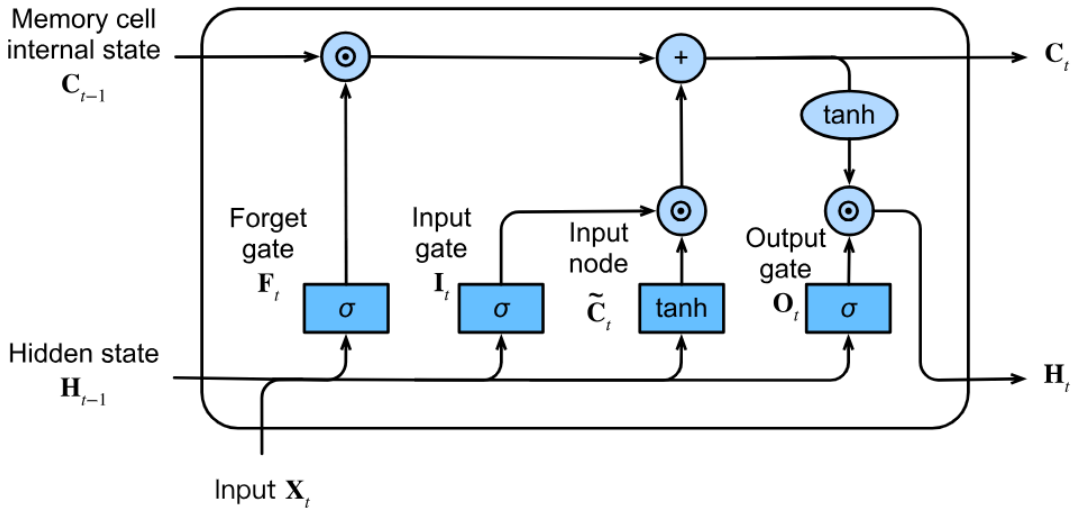


**Figure 2.4:** Structure of an LSTM cell showing the forget, input, and output gates that regulate the flow of information between time steps [2].

As illustrated in the figure, the previous cell state $C_{t-1}$ and the hidden state $H_{t-1}$ are combined with the input $X_t$ to compute the gate activations $F_t, I_t, O_t$, forget, input, and output, respectively. The input node, denoted $\tilde{C}_t$ acts as a draft for the new information and determines, together with the input gate, what new information

should be written to the cell state $C_t$.

LSTMs can be leveraged within the autoencoder architecture to encode sequences of observations into latent space while capturing the temporal dependencies that are apparent in time-series data [38, 39, 40, 41].

While recurrent architectures such as LSTMs have been widely used for time-series analysis, the sequences are processed step-by-step, which can make training computationally expensive. Transformer-based architectures have grown exponentially in popularity ever since the emergence of LLMs and have been appropriated for many different tasks, among them time-series analysis [42, 43]. One of the main benefits of transformers, compared to recurrent architectures, is that they are able to capture relationships in a sequence without processing the inputs sequentially.

#### 2.2.3.2 Transformer Autoencoders

The core component of a transformer is the *self-attention* mechanism. Originally designed to be used in encoder-decoder RNNs for sequence modelling tasks [2], self-attention enables the model to weigh the importance of different observations within the sequence when computing the representation of a given time step. Each part of the sequence can be provided as context for constructing representations of other parts; the weights are then assigned based on how important this feature is deemed for computing the specific observation [3].

Since the attention mechanism allows for the model to process a sequence without propagating it through many sequential layers, it can capture long-range dependencies more effectively than recurrent models [2]. The architecture of the transformer is also more suitable for parallelisation, as attention computations can be performed using matrix operations over an entire sequence. Matrix operations are heavily optimised on GPUs compared to CPUs, leading to further improvements in efficiency and decreased computational complexity.

Figure 2.5 illustrates the transformer architecture. Since transformers process inputs simultaneously rather than sequentially, a positional encoding is added for each input in order to preserve the temporal ordering of observations. The transformer consists of a stack of encoder and decoder layers. An example of an encoder layer is shown to the left in Figure 2.5, and the decoder is illustrated to the right. The $N\times$ notation indicates that the layer is repeated $N$ times in the stack.

Each layer in both encoder and decoder stacks contains two main components: a *multi-head self-attention* mechanism and a feed-forward network. Self-attention in this architecture is "multi-head", meaning that instead of computing one attention pattern, the transformer computes multiple (each one referred to as a head) in parallel. The heads learn different relations between observations, for example, one can focus on short-term dependencies while another one computes correlations between variables. These heads are then combined, allowing the transformer to capture different types of relationships in parallel.

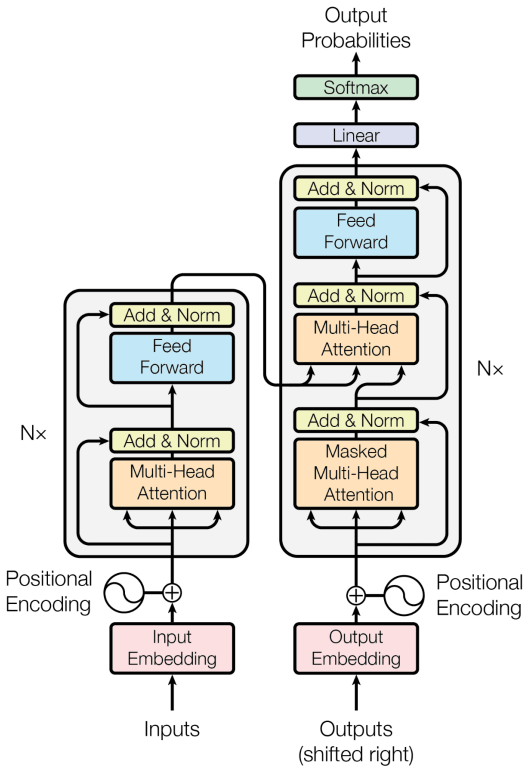


**Figure 2.5:** Overview of the transformer architecture consisting of an encoder and decoder stack. Each layer contains multi-head self-attention and feed-forward components combined with residual connections and normalisation. Positional encodings are added to preserve the ordering of sequence elements [3].

### 2.2.4 Applications of autoencoders

Autoencoders offer two key advantages to traditional dimensionality reduction techniques:

- **Non-linear compression**: The encoder can capture relations between features that are not linearly separable

- **Dual functionality of representations**: The latent representation $z$ learned by the encoder provides a compact feature representation that can be used for downstream analysis tasks such as clustering, similarity search, or visualisation of high-dimensional data. At the same time, the reconstruction error between the input $x$ and its reconstruction $\hat{x}$ can be used as an anomaly score. Since the autoencoder is trained to reconstruct patterns that occur in the training data, inputs that deviate from these learned patterns typically produce larger reconstruction errors, which can indicate anomalous behaviour [11, 44].

Autoencoders have been successfully applied for network intrusion detection [11], and log anomaly detection tasks [7, 12], demonstrating their suitability for high-dimensional time-series data analysis.

## 2.3 Threshold Estimation for Anomaly Detection

Reconstruction-based anomaly detection methods assume that a model trained exclusively on normal, non-anomalous data, learns a compact representation of the normal system behaviour. During inference, the difference between the input and its reconstruction can therefore be interpreted as an anomaly score, where larger reconstruction errors indicate a greater deviation from the learned normal behaviour [34]. A threshold $\tau$ on the reconstruction error can be applied to classify observations as either normal or anomalous. The choice of threshold is critical, as excessively low thresholds may lead to large numbers of false positives, while excessively high thresholds may fail to detect true anomalies.

A simple and commonly used approach is empirical percentile thresholding, where the threshold is selected as a high percentile of the reconstruction error distribution [45, 46]. Under the assumption that normal observations constitute the majority of the data, most reconstruction errors are expected to form the bulk of the distribution, while anomalous observations appear in the upper tail. For example, selecting the $95^{\text{th}}$ percentile as the threshold implies that the largest 5% of reconstruction errors are classified as anomalous. Although percentile-based thresholding is computationally simple and easy to implement, it relies directly on the empirical distribution of observed errors and may therefore generalise poorly when the tail behaviour differs between datasets.

To obtain a more principled characterisation of extreme reconstruction errors, methods from Extreme Value Theory ($EVT$) can instead be used to model the tail of the error distribution [47]. EVT is a branch of statistics concerned with the probabilistic modelling of rare or extreme events. Rather than modelling the complete distribution, EVT focuses specifically on the asymptotic behaviour of observations in the distribution tail. This makes EVT particularly suitable for anomaly detection, where anomalies are typically associated with unusually large reconstruction errors [48].

One common EVT formulation is the Peaks-Over-Threshold ($POT$) method [49]. In POT, a sufficiently high preliminary threshold $u$ is selected, and only observations exceeding this threshold are considered. The exceedances are defined as

$$Y = X - u \mid X > u$$

where $X$ denotes the reconstruction error and $u$ is the selected threshold. Under mild regularity conditions, the distribution of exceedances converges asymptotically to a Generalised Pareto Distribution ($GPD$) [47]. The cumulative distribution function of the GPD is given by

$$G(y; \xi, \beta) = 1 - \left(1 + \frac{\xi y}{\beta}\right)^{-1/\xi}$$

where $\xi$ is the shape parameter controlling tail behaviour and $\beta > 0$ is the scale parameter.

In time-series anomaly detection, reconstruction errors are commonly computed over sliding windows. Since adjacent windows often share a large proportion of their observations, neighbouring anomaly scores become strongly temporally correlated. Classical POT-based EVT methods assume approximately independent exceedances, and directly fitting a GPD to highly overlapping scores may therefore violate these assumptions. A common approach for reducing temporal dependence is declustering, where correlated exceedances are grouped together and represented by local maxima separated by a minimum temporal distance. This produces a set of approximately independent extreme observations suitable for EVT modelling.

## 2.4 Metrics for Binary Classification

True and false positives are outcomes of a binary classification task based of how the predicted label of an observation compares to its true label, together with false positives and false negatives they model all the possible outcomes of a binary classification task. These four outcomes forms the *confusion matrix*, which summarises the relationship between the predicted and true labels across the dataset. In the context of anomaly detection, where the positive class typically corresponds to anomalous observations, the four outcomes can be described as:

- **True Positive (TP)**: an anomalous observation correctly classified as anomalous.
- **True Negative (TN)**: a normal observation correctly classified as normal.
- **False Positive (FP)**: a normal observation incorrectly classified as anomalous.
- **False Negative (FN)**: an anomalous observation incorrectly classified as normal.

To evaluate the effectiveness of the anomaly detection model, standard evaluation metrics such as precision, recall, and F1-score are used.

Precision measures how well the model predicts the positive predictions and is defined as

$$\frac{TP}{TP + FP}$$

Recall measures the quantity of the positive detections and is defined as

$$\frac{TP}{TP + FN}$$

F1-Score acts like a balance between precision and recall and is defined as

$$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Matthews Correlation Coefficient (MCC) is a classification metric that evaluate prediction quality using all parts of the confusion matrix. An advantage of MCC, as compared to F1-score, is that it remains informative for highly unbalanced datasets by incentivising the prediction model to perform well on both the majority and minority classes in order to achieve a high score. This is particularly suitable in anomaly detection tasks where a majority of the observations are non-anomalous. Formally, MCC is computed by:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

The MCC score ranges from $-1$ to 1, where 1 represents perfect classification, 0 corresponds to random prediction, and $-1$ indicates complete disagreement between predictions and ground truth labels[50].

## 2.5 Clustering

Beyond summarisation, clustering techniques have been used to identify structure in high-dimensional time-series data, enabling the discovery of recurring patterns and anomalies that are difficult to detect by manual inspection [14].

Clustering is an unsupervised learning technique used to organise datapoints into groups, or clusters, such that points within the same cluster are more similar to each other than to those in different clusters [51]. While similarity or distance are traditional metrics for clustering, there also exist other paradigms such as density-based or model-based clustering [10, 52]. Given a dataset $X = \{x_1, \ldots, x_n\}$, the result of a clustering algorithm can be seen as a clustering assignment function

$$c : X \to \{1, \ldots, K\}$$

that maps each datapoint to a cluster label indicating which group it belongs to. Some clustering algorithms allow datapoints to remain unassigned to any cluster, which is useful when handling noise or anomalies [52]. This can be represented by introducing an additional label for all unassigned datapoints $x_i$ s.t. $c(x_i) = 0$. Since clustering can be done without any predefined labels, it is referred to as an unsupervised learning task [51].

### 2.5.1 K-Means

K-means is the most widely used centroid-based clustering algorithm. In this framework, data points are partitioned into clusters by assigning each point to the nearest centroid in the feature space. A centroid is defined as the geometric mean of all points within a cluster. The notion of nearest is determined by a chosen distance metric; in k-means, the squared Euclidean distance is most commonly used. The algorithm iteratively updates cluster assignments and centroids to minimise the within-cluster variance [53].

The clusters formed by the k-means algorithm are convex, isotropic, and roughly equal in size, since assignments are based on the Euclidean distance to the nearest centroid. K-means relies on these implicit assumptions about the structure of the data, which makes it well-suited for clearly separated, spherical clusters, but less effective when clusters have irregular shapes, varying densities, or overlap in feature space. Furthermore, k-means performs hard assignments of each data point to a single cluster, meaning that it cannot represent uncertainty in cluster membership or ambiguity between clusters.

### 2.5.2 Fuzzy C-Means

A popular extension of k-means that relaxes the hard assignment of points to clusters is *fuzzy c-means*, where each datapoint belongs to clusters with varying degrees of membership rather than belonging to a single cluster exclusively. For each point, the fuzzy c-means algorithm assigns a membership value between 0 and 1 for each cluster such that the values of membership functions for each datapoint sum to one. This allows for a point to belong to multiple clusters of varying degree while also having a system that can collapse into a discrete cluster assignment.

The algorithm still relies on cluster centroids as in the traditional k-means method, but both the centroid updates and the assignment step are weighted based on the fuzzy membership values. Points that strongly belong to a certain cluster have a larger influence on its centroid while points that don't contribute proportionally less and distribute their influence across multiple clusters. This property makes fuzzy c-means clustering better suited in data spaces where cluster structure is not well separated or with gradual transitions between regimes [54].

### 2.5.3 DBSCAN

In contrast, DBSCAN (Density-Based Spatial Clustering of Application with Noise) defines clusters as dense regions separated by sparse areas, which allows it to identify arbitrarily shaped clusters and to label isolated points as noise [52]. DBSCAN works by grouping datapoints based on "density connectivity". This means that, instead of requiring each point within a cluster to be within a specific distance of every other point, two points are considered connected if there is a "bridge-point" (or multiple in a series) sufficiently close to each of them forming the connection. Thanks to this property, DBSCAN allows for any set of points that together form a continuous dense region to form a cluster. An example of how DBSCAN can create these arbitrarily shaped clusters and how k-means compares can be seen in Figure 2.6.

Since DBSCAN does not require each point to be part of a cluster (as in k-means), it is able to process outliers by marking points that do not belong to any sufficiently dense neighbourhood as noise. These properties make DBSCAN particularly suitable for datasets with irregular cluster structures and in settings where noise points are of direct analytical interest, such as anomaly detection tasks [52].

Traditional clustering methods work well for small to moderately sized datasets

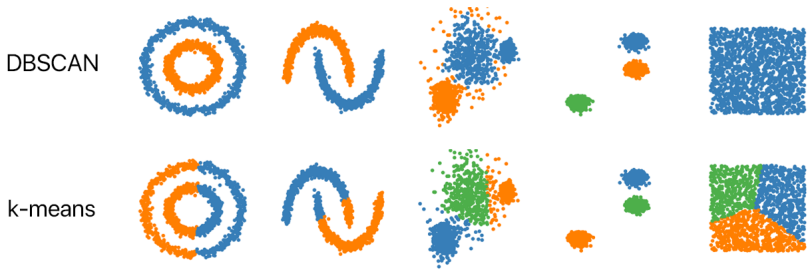


**Figure 2.6:** DBSCAN and k-means difference in resulting cluster over complex shapes [4]

but struggle with high-dimensional and large-scale data due to the computational cost of repeatedly computing distances between points [14]. To combat this, recent work has combined clustering techniques with Locality-Sensitive Hashing (LSH), an approximate algorithm for nearest-neighbour search, reducing the cost of similarity queries [55]. The general idea behind LSH is to hash items using a variety of hash functions. These different functions all have the property that they map similar items to the same hash bucket with high probability. Candidate pairs of similar items are those that end up in the same hash bucket for at least one of the functions.

Previous work demonstrates that clustering pipelines can efficiently process large temporal data by combining dimensionality reduction with a clustering method like DBSCAN, and similarity search, via LSH, making them well-suited for grouping high-dimensional data too large or complex to analyse directly [13, 14]. Keramatian et al. [14] shows that LSH can be used to narrow down which datapoints are likely to be close to each other in the cluster, providing the DBSCAN algorithm with a smaller candidate set of points to compute distances with instead of comparing all points with each other. The candidate set for a given point is built by the points that hash to the same buckets by LSH. This drastically reduces the amount of distance computations required for each step of the algorithm and makes DBSCAN clustering feasible on high-dimensional data. Building on this, the CLUE toolchain [13] demonstrates how LSH-accelerated density-based clustering can be implemented in practice, enabling identification of consumption patterns and anomaly behaviours in real-world electrical grid data [13].

While IP-LSH-DBSCAN addresses the computational efficiency of distance calculations, the quality of the resulting clusters is highly dependent on the quality of the underlying data. Traditional methods, like DBSCAN, can struggle to capture non-linear relations in high-dimensional data [10]. Recent research in representation learning-based clustering shows that transforming raw time-series into a lower-dimensional latent space enables the model to capture temporal dependencies and semantic similarities inherent in high-dimensional data but often masked by noise

in raw signals [10]. Combining representation learning with clustering methods like DBSCAN can, in turn, improve clustering performance compared to other summarisation methods, as the learned embeddings can capture more complex relations [9, 10, 56].

The behaviour and performance of clustering algorithms are in large dependent on their notion of similarity used when comparing datapoints. Distance metrics determine how similarity is quantified between pairs of points in the feature space, and thus the choice of distance metric has a strong influence on the resulting clustering assignments.

### 2.5.4 Distance Metrics

As mentioned earlier, Euclidean distance is the most common distance metric for the k-means and fuzzy c-means algorithms. This is largely due to the way it measures similarity: it computes the geometric distance between points and therefore implicitly assumes that clusters are compact and spherical in shape. For two points $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_n)$ in an $n$-dimensional space, the Euclidean distance can be computed by:

$$dist(P, Q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

The result is bounded below by 0 and unbounded above, meaning it lies in the range $[0, \infty)$ [30].

An alternative similarity measurement is cosine similarity, which is often used when the directional relationship between vectors is more important than their magnitude. Cosine similarity measures the cosine of the angle between two vectors by computing the dot product between the two before normalising by the product of their magnitudes. Using the same $P$ and $Q$ as before:

$$sim(P, Q) = \frac{P \cdot Q}{\|P\| \, \|Q\|}$$

The similarity ranges from $-1$ to 1, where $-1$ indicates opposite direction and 1 indicates identical direction, a score of 0 means the vectors are orthogonal and unrelated. Cosine similarity is often more informative in latent representations where meaningful structure is encoded in the orientation of vectors rather than their scale [30].

Euclidean distance is sensitive to the scale and magnitude of the data, as features with high variance or large magnitude contribute disproportionately to the resulting distance. Due to this, feature normalisation is an important preprocessing step when using Euclidean-based distance methods. In contrast, cosine similarity measures the angle and is thus less sensitive to different magnitudes in feature values. Returning to our previous example and setting $n = 2$ and the vectors $P = (1, 1)$ and $Q = (100, 100)$, the Euclidean distance would compute to $dist(P, Q) \approx 140$ while the cosine similarity is $sim(P, Q) = 1$.

# 3

# Problem

While representation learning offers a promising approach to analysing high-dimensional time-series data, two key questions remain open: whether learned representations can serve both anomaly detection *and* meaningful clustering of those anomalies, and whether increased model complexity actually translates into better downstream analysis compared to simpler methods. This chapter narrows these questions into a concrete problem formulation, motivated by an industrial case study.

## 3.1 Industrial Motivation: Ericsson Case-Study

To ground this investigation in a concrete industrial setting, this work includes a case study conducted in collaboration with Ericsson. The case study examines large-scale time-series data generated during continuous integration and continuous deployment (CI/CD) testing of network components.

The testing pipeline generates thousands of performance metrics (PMs) from network components during automated test executions. A test failure is typically observed as one or more PMs violating a threshold or deviating from an expected range. However, the underlying *causes* of these failures are not directly observable from the violations alone. Since the test logic does not explicitly alter the PMs, determining the drivers of a failure is a complex manual task. Furthermore, failures originating from the same drivers may recur across multiple test executions while exhibiting different observable symptoms. Figure 3.1 shows an example test case run where multiple assertions fail over time.

This pipeline consists of two main components: the *Traffic Generator* and the *Controller*. The *Traffic Generator* simulates user equipment and generates network traffic, while the *Controller* serves as the central component responsible for processing and orchestrating that traffic. Logging is performed across many different components in the pipeline, with each component producing several types of logs. However, this study focuses exclusively on the *Controller* component and, more specifically, on its PMs, such as memory and CPU usage. This focused scope enables detailed analysis but does not capture full system behaviour, precluding comprehensive root cause analysis.

This setting illustrates the core problem concretely: individual test case runs may span several days and contain up to ten thousand PMs, resulting in high-dimensional and large-scale datasets. In addition, PMs may appear in multiple variants depending on how the system is set up. As a result, treating each observed variant as a distinct feature can substantially increase the effective dimensionality of the data.

Engineers need tools that can (1) detect when system behaviour deviates from normal operation, (2) group similar deviations together regardless of when or in which test run they occur, and (3) characterise what makes each group distinct, enabling faster diagnosis of recurring faults.

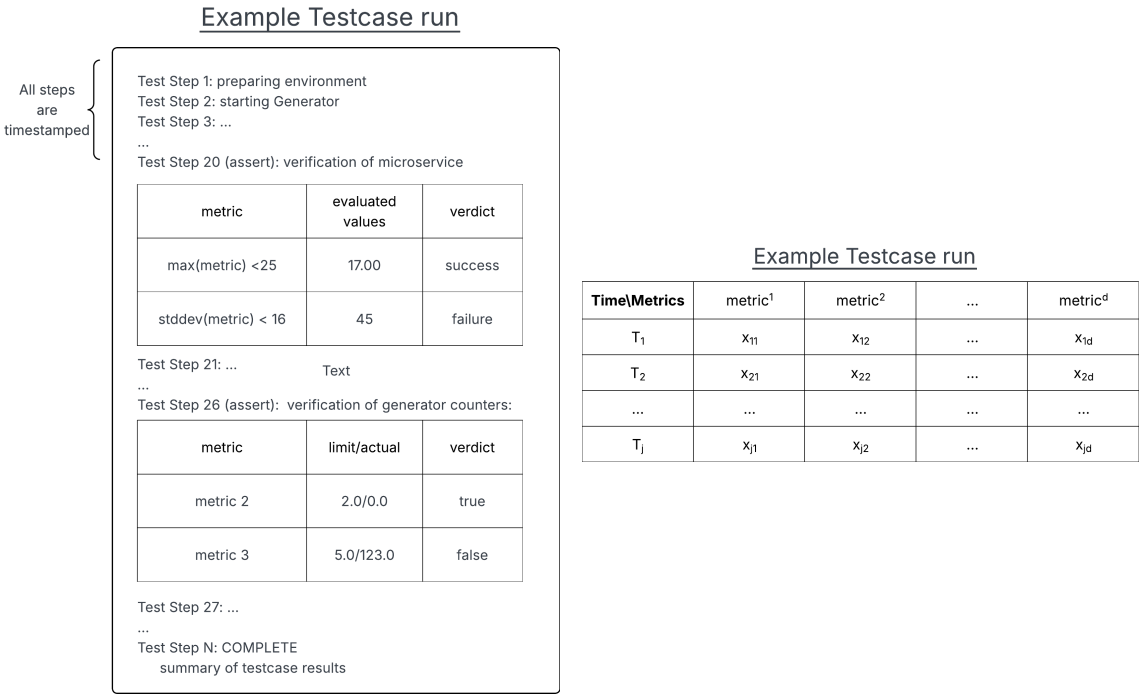


**Figure 3.1:** Example output of one test run (left) and its formal representation as a multivariate time-series (right).

Ericsson Research is also a part of the Wallenberg AI, Autonomous Systems and Software Program (WASP) Research Arena (RA) *WARA-Ops*, where one of the main research focus areas is anomaly detection [57].

## 3.2 Research Questions

This thesis investigates whether learned representations can produce stable, interpretable, and diagnostically useful groupings of anomalous behaviour in high-dimensional time-series data. Specifically, we address the following research questions:

**RQ1: Reconstruction and anomaly detection.** How well can different representation learning methods reconstruct normal system behaviour, and how

effectively can reconstruction error distinguish anomalous from normal observations? Do complex deep learning architectures offer meaningful improvements over traditional dimensionality reduction techniques?

**RQ2: Stability.** Do the learned representations produce consistent groupings under perturbation? If the input data is slightly perturbed (e.g., by noise, resampling, or model reinitialisation), do the resulting cluster assignments remain stable, or do they instead reflect noise and randomness?

**RQ3: Interpretability.** Can the resulting groupings be explained in terms of distinct patterns in the original feature space? Do different clusters correspond to different subsets of features or different types of system behaviour, enabling engineers to understand *why* observations are grouped together?

**RQ4: Computational trade-offs.** How do the evaluated methods compare in terms of computational cost (training time, inference latency, memory usage) relative to their representational quality? Does increased model complexity justify its cost?

## 3.3 Desired Properties of a Solution

A solution to the stated problem must satisfy several properties. These are defined conceptually here while the metrics used to assess them are presented in Chapter 5.

### 3.3.1 Representation Quality

The learned representation must compress high-dimensional multivariate time-series windows into a compact latent vector while preserving the information necessary for both anomaly detection and downstream clustering. The representation should capture temporal patterns, inter-feature correlations, and characteristic system behaviours that distinguish normal operation from anomalous events.

### 3.3.2 Anomaly Sensitivity

The representation must enable discrimination between normal and anomalous observations. Since the models are trained exclusively on normal data, observations that deviate from learned patterns should produce measurably larger reconstruction errors, providing a basis for anomaly scoring.

### 3.3.3 Stability

The groupings produced by clustering the learned representations should be robust to small perturbations in the input data, the model initialisation, or the sampling of observations. Unstable groupings that change substantially under minor variations cannot be trusted for diagnostic purposes and likely reflect noise rather than genuine structure.

### 3.3.4 Interpretability

The discovered groups should be explainable in terms of the original features. Ideally, each cluster should correspond to a distinct behavioural pattern. As an example, one cluster could be characterised by CPU-related anomalies and another by memory-related anomalies. This requires that the latent space preserves enough structure to produce clusters that map back to meaningful distinctions in the input space.

### 3.3.5 Diagnostic Usefulness

Beyond detection, the system should support diagnosis by grouping anomalous observations according to their underlying behaviour, regardless of when or in which test run they occur. This enables engineers to identify recurring failure types, relate new anomalies to previously observed patterns, and reason about classes of failures rather than individual instances.

## 3.4 Scope and Delimitations

This section defines the boundaries of the investigation. Limitations encountered *in practice* during the work are discussed in Section 5.7.

- **Supervised and semi-supervised methods** are not used during model training. Labelled data, where available, is used only for evaluation and threshold calibration.

- **Real-time and online detection** is not addressed. All experiments are conducted in an offline batch setting; latency and deployment constraints in streaming environments are not explored.

- **Architecture optimisation** is not the goal. Multiple autoencoder architectures are compared, but the focus is on analysing trade-offs between representation quality and computational efficiency rather than exhaustive model tuning.

- **Root cause identification** is not claimed. The work aims to group similar anomalies to *support* root cause analysis, but does not perform causal inference or automated diagnosis.

- **Production deployment** is not addressed. The emphasis is on evaluating representation quality and clustering effectiveness, not on building production-grade pipelines or operational systems.

The evaluation is conducted on established benchmark datasets (SMD, SWaT, SMAP) that provide labelled anomalies for quantitative assessment, as well as the unlabelled industrial dataset from Ericsson for scalability and qualitative evalua-

tion. This combination allows both rigorous comparison against ground truth and assessment of practical applicability at industrial scale.

# 4

# Methodology

This chapter formalises the problem setup, outlines the approach taken to address the research questions, and discusses key challenges associated with the problem, and the reason behind the chosen methods.

## 4.1 Formal Setup

We begin by formalising the input and the representation learning task.

We consider a set of multivariate time-series $S = \{s_1, \ldots, s_m\}$, of different lengths $T_j$, where each $s_j \in \mathbb{R}^{d \times T_j}$ represents $d$ features observed over a time horizon $T_j$. This formulation allows $S$ to consist of multiple multivariate time-series while reducing to a single one in the case of $m = 1$.

To enable analysis of large-scale multivariate time-series, we extract fixed-length windows of length $T$ from these time-series. Each window is therefore represented as $x_i \in \mathbb{R}^{d \times T}$, containing $d$ features observed over $T$ consecutive time-steps. We extract $n$ windows and can then define the dataset used for representation learning as

$$X = \{x_1, \ldots, x_n\}.$$

The goal of representation learning is to learn a non-linear embedding function

$$f_\theta : \mathbb{R}^{d \times T} \to \mathbb{R}^k$$

that maps each time window $x_i$ to a latent representation

$$z_i = f_\theta(x_i), \qquad z_i \in \mathbb{R}^k$$

where the latent dimension $k$ is chosen such that $k \ll (d \times T)$. The resulting set of latent representations is denoted by

$$Z = \{z_1, \ldots, z_n\}.$$

The representation $z_i$ must then condense the information along the temporal axis ($T$) and between the features ($d$) of the original window into a fixed-length ($k$) vector that can later be used for analysis.

Example

As a running example, consider a monitored computing system that tracks system metrics such as CPU utilisation, memory usage, and network throughput over time. Within a short time window, different types of anomalous behaviours may produce characteristic patterns across these metrics. For example, a sudden spike in CPU usage may coincide with increased memory activity or network traffic. A useful representation should preserve these patterns so that windows exhibiting similar anomalous behaviours are mapped to nearby points in the latent space.

## 4.2 Approach

To investigate whether learned representations can produce stable, interpretable, and diagnostically useful groupings of anomalous behaviour in high-dimensional time-series data, a representation learning approach based on autoencoders will be implemented. The model is trained to compress high-dimensional data into a lower-dimensional latent representation and reconstruct the original input from this representation. The reconstruction error between the original and reconstructed datapoint is used for anomaly detection. Observations where the error exceeds a certain threshold will be considered anomalies, the threshold is chosen using the Peak-Over-Threshold (POT) method described in Section 2.3.

This approach will be evaluated in comparison with traditional dimensionality reduction techniques, such as PCA and random projections, introduced in Section 2.1.

The outlined approach consists of the following steps and will be the same for the different implementations of autoencoder architectures:

- Implement autoencoder architecture
- Train an autoencoder on normal system behaviour to learn a compact representation of the input data. We say we have learned the compact representation when $\theta$ and $\phi$ converge (see 2.2.1.1).
- Detect anomalous observations by measuring reconstruction error between original input data and the reconstructed output (see 5.1.1).
- Cluster found anomaly-representations by the algorithms mentioned in the background (see 2.5)

For clustering, density-based approaches are suitable in practice as they do not require the number of clusters to be specified in advance, and outliers are naturally identified, while k-means clustering is common in previous academic work [56, 58, 59].

## 4.3 Evaluated Models

The following models are evaluated and compared. They span traditional dimensionality reduction techniques and several deep learning architectures designed for temporal data.

### 4.3.1 Traditional Baselines

- **PCA** Principal Component Analysis. Linear projection onto directions of maximum variance (Section 2.1.1).
- **KPCA** Kernel PCA. Non-linear extension of PCA using a kernel function to capture non-linear structure (Section 2.1.3).
- **RP** Random Projections. Linear projection using a random matrix that approximately preserves pairwise distances (Section 2.1.2).

### 4.3.2 Autoencoder Architectures

- **AE** Standard feedforward autoencoder. Serves as the baseline neural network architecture (Section 2.2.1).
- **VAE** Variational Autoencoder. Regularises the latent space toward a standard normal distribution via KL-divergence (Section 2.2.2).

### 4.3.3 Temporal Autoencoder Architectures

- **LSTM** LSTM-based recurrent autoencoder. Captures temporal dependencies through gated recurrent cells (Section 2.2.3.1).
- **TransAE** Transformer-based autoencoder. Uses self-attention to model long-range temporal relationships (Section 2.2.3.2).

### 4.3.4 Specialised Anomaly Detection Architectures

- **USAD** Dual-decoder autoencoder with adversarial training to improve separation between normal and anomalous behaviour [60].
- **DAGMM** Deep Autoencoding Gaussian Mixture Model. Combines reconstruction with density estimation in the latent space [61].
- **TranAD** Transformer-based architecture with multi-step reconstruction and attention-based anomaly amplification [42].
- **OmniAnomaly** Stochastic recurrent VAE with GRU-based temporal modelling and regularised latent space [38].

All methods follow the same pipeline: the encoder (or projection) maps input windows $x_i \in \mathbb{R}^{d \times T}$ to latent representations $z_i \in \mathbb{R}^k$, and the decoder (or back-projection) reconstructs $\hat{x}_i$ from $z_i$. The reconstruction error $R(x_i)$ serves as the anomaly score. For clustering, the latent representations $Z$ are used directly or after further embedding via UMAP or t-SNE. This uniform pipeline enables fair comparison across all methods.

## 4.4 Challenges

The problem of detecting anomalies, and grouping structural behaviour in high-dimensional multivariate time-series data presents several challenges. These challenges motivate the choices of the proposed approach.

For anomaly detection, information retention is crucial. As for all summarisation techniques, choices on aggregations and summarisation methods that distinguish normal from abnormal behaviour are difficult. In summarisation-based approaches, choices such as normalisation, latent dimensionality and loss function formulation directly affect whether anomalies can be detected or will be indistinguishable from other behaviour and noise. To address this, the choice of autoencoders is made, which learns a compact latent representations by minimising the reconstruction error. The compact latent representation forces the model to learn important structural and temporal patterns instead of modelling noise. Autoencoders also have the advantage of being able to learn complex non-linear relationships between different time-series which PCA and Random Projections are not suited for.

A challenge related to anomaly detection is defining similarity between anomalies in a way that enables meaningful clustering. Anomalous behaviours in systems may vary in duration, magnitude, and structure, and datasets are often highly imbalanced with few total anomalies and heterogeneous patterns [62, 63]. The proposed approach addresses this by mapping the input data into a latent space, where similar patterns are expected to be encoded closer together. Clustering is then performed in this space where the data is represented in a compact and normalised vector form, making similarity comparisons robust.

Another interesting practical issue arises when working with anomalies and high-dimensional data. A relatively small classification or reconstruction error in a subset of the data, or in a smaller dataset, can have great effects when the output is used for grouping at a larger scale. For example, a low false positive rate measured per run can create large amounts of anomaly instances when a larger amount of data is processed. This could lead to misleading clusters and make similarity search more difficult, as false anomaly instances introduce noise into the data. Further, it could mean that performance achieved at a small scale does not necessarily reflect the behaviour in the industrial case study. To account for this, we include evaluation on both benchmark datasets and large-scale industrial data, allowing assessment of how error behaviour propagates and affects downstream clustering in practice. Further information of the datasets studied in this thesis can be found in Section 5.2.

In the industrial case-study, scalability and computational constraints are other practical obstacles. Methods that work well on benchmark datasets may become infeasible when used on the very large industrial data. This creates trade-offs between model complexity, inference time, memory usage and granularity of summarisation. This is because more expressive models and higher-dimensional latent representations can capture finer details in the data, but require increased computational resources and result in higher storage and processing costs during both training and downstream analysis. For example, higher-dimensional latent spaces may improve expressiveness but increase storage and search cost. The proposed approach explicitly considers these trade-offs by comparing different autoencoder architectures and evaluating their computational efficiency alongside their representation quality.

Stability, interpretability and usefulness all come with challenges from a user point of view. Latent representations and cluster assignments must be robust to small perturbations if they are to be interpretable to domain experts and useful for diagnosing behaviours. However, latent representations are inherently abstract, and clustering results may be sensitive to both variations in data and the model. As the data in the case-study is unlabelled, evaluating these properties becomes particularly challenging. To address this, the evaluation framework focuses on stability metrics, clustering validation measures and qualitative assessments through the case-study.

Another challenge that arises in all data, both labelled benchmark data and industrial, is data quality. Poor data quality complicates both representation learning and clustering. Preprocessing decisions, such as resampling and feature selection, interact with model behaviour and can introduce biases that affect anomaly detection and clustering quality. We acknowledge this by ensuring comparisons between methods are not influenced by differences in data treatment. While the project relies on the provided datasets and does not attempt to fix data quality issues, this controlled setup allows for fair comparisons and evaluation of the proposed approach.

# 5

# Evaluation

This chapter evaluates the representation learning methods introduced in Chapter 4 against the research questions defined in Chapter 3. Section 5.1 defines the metrics and protocols used for assessment. Sections 5.2–5.4 describe the experimental setup, and Section 5.5 presents the results.

## 5.1 Evaluation Goals and Methodology

Reconstruction quality assesses how well the produced latent representations preserve the structure of the original input data. This directly addresses RQ1: whether the methods can reconstruct normal behaviour and whether reconstruction error can distinguish anomalous from normal observations. From reconstruction quality we derive the anomaly score used for detection. Sections 5.1.1–5.1.2 define the metrics and protocols for this assessment, while Section 5.1.4 addresses RQ2 (stability), Section 5.1.5 addresses RQ3 (interpretability), and Section 5.1.6 addresses RQ4 (computational trade-offs).

### 5.1.1 Reconstruction Metrics

Reconstructing the original input from the latent representation is a well-studied issue [1, 11]. It can work as an indirect method for measuring the quality of dimensionality reduction and its ability to retain important information. To measure reconstruction quality, one often measures the error between the original input data and the data reconstructed from its latent representation. This is usually evaluated by three complementary metrics that are defined below.

Let $X = \{x_1, \ldots, x_n\}$ denote the dataset of time windows, where each $x_i \in \mathbb{R}^{d \times T}$, and let $\hat{x}_i$ denote the reconstruction of $x_i$. When referring to a generic window, we omit the index $i$ and write $x \in \mathbb{R}^{d \times T}$.

#### 5.1.1.1 Mean Squared Error (MSE)

MSE measures the average squared deviation between the original input and its reconstruction. For a window $x \in \mathbb{R}^{d \times T}$, the mean squared error is defined as

$$\mathrm{MSE}(x, \hat{x}) = \frac{1}{dT} \sum_{j=1}^{d} \sum_{t=1}^{T} (x_{jt} - \hat{x}_{jt})^2 .$$

For notational convenience, this can equivalently be written using the Frobenius norm as

$$\mathrm{MSE}(x, \hat{x}) = \frac{1}{dT} \|x - \hat{x}\|_F^2,$$

where the Frobenius norm of a matrix $A \in \mathbb{R}^{d \times T}$ is defined as

$$\|A\|_F = \sqrt{\sum_{j=1}^{d} \sum_{t=1}^{T} |A_{jt}|^2}.$$

The average reconstruction error over the dataset is then

$$\frac{1}{n} \sum_{i=1}^{n} \mathrm{MSE}(x_i, \hat{x}_i).$$

#### 5.1.1.2 Average Per-Feature Normalised Reconstruction Error

An extension of MSE that accounts for differences in feature scale is *average per-feature normalised reconstruction error*. This metric evaluates reconstruction quality relative to the variance of each feature.

For a dataset $X = \{x_1, \ldots, x_n\}$, let $\bar{x}_j \in \mathbb{R}^T$ denote the mean time-series of feature $j$ across all windows, i.e.

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij},$$

where $x_{ij} \in \mathbb{R}^T$ denotes the time-series corresponding to feature $j$ in window $x_i$.

The normalised reconstruction error is then defined as

$$\frac{1}{d} \sum_{j=1}^{d} \frac{\frac{1}{n} \sum_{i=1}^{n} \|x_{ij} - \hat{x}_{ij}\|_2^2}{\frac{1}{n} \sum_{i=1}^{n} \|x_{ij} - \bar{x}_j\|_2^2}.$$

#### 5.1.1.3 Relative Frobenius Norm Reconstruction Error

Another commonly used reconstruction metric is the *Relative Frobenius norm reconstruction error*, which measures the overall deviation between the original data and the reconstructed data.

Let $X \in \mathbb{R}^{n \times d \times T}$ denote the collection of all windows and $\hat{X}$ their reconstructions. The relative reconstruction error is defined as

$$\frac{\|X - \hat{X}\|_F}{\|X\|_F},$$

where the Frobenius norm is computed over all entries in the dataset.

### 5.1.2 Reconstruction Error Evaluation

Given a model trained on non-anomalous data, assume it will reconstruct the normal instances well, while anomalous instances will yield a higher reconstruction error. Let $R(x)$ denote a reconstruction error function (e.g. MSE). An anomalous instance is one where $R(x_i)$ exceeds some predefined threshold $\tau$ i.e. $R(x_i) > \tau$.

The threshold $\tau$ is applied to classify observations as normal or anomalous. The value of $\tau$ is a trade-off between detecting true anomalies and avoiding false positives and can be tuned during training using validation data or by analysis of the distribution of reconstruction errors.

However, directly choosing a threshold based on reconstruction error can be sensitive to the dependence introduced by the overlapping windows. Since overlapping windows share a large part of their observation (note that the extreme case of stride being one shares all datapoints except the first and last), the resulting anomaly scores are strongly temporally correlated, which does not satisfy the approximate independence assumptions required by classical Extreme Value Theory (EVT) [47, 48].

This problem is addressed by applying a declustering step before the threshold estimation, which extracts locally maxima scores separated by a minimum temporal distance corresponding to the structure of the overlapping windows. This yields a reduced set of approximately independent exceedances, making it more suitable for tail modelling.

After declustering, $\tau$ is estimated using the Peaks-Over-Threshold (POT) framework from EVT [49], mentioned in Section 2.3. A high empirical quantile is first selected as a preliminary threshold $u$ and exceedances above this level are modelled using a Generalized Pareto Distribution (GPD). The fitted tail model is then used to extrapolate beyond the empirical distribution and estimate a threshold corresponding to a desired false positive rate $\alpha$. In cases where the number of exceedances are too few for a stable tail estimation, a fallback percentile-based threshold is used instead.

### 5.1.3 Evaluation Granularity

Anomaly detection in time series can be evaluated at different granularities. At the *window level*, each sliding window receives a single binary label and the classification metrics are computed over windows directly. At the *pointwise level*, window-level scores are broadcast back to individual time steps (e.g. by taking the maximum score

among all windows covering a given point), and metrics are computed per time step against ground-truth point labels.

A third strategy common in the literature is *point-adjusted* (PA) evaluation [38, 60, 64], in which a detected anomaly at *any* point within a contiguous anomalous segment counts as detecting the *entire* segment, i.e. all points in that segment are counted as true positives. While PA evaluation was originally motivated by the argument that alerting an operator once per event is sufficient, it has been shown to severely inflate recall and F1-scores, since a single fortunate detection can mask an otherwise poor detector [64]. Moreover, PA evaluation conflates detection *timeliness* with detection *quality*: a method that flags only the last point of a long anomalous segment receives the same credit as one that detects it immediately. For these reasons, we adopt strict pointwise and window-level evaluation without point adjustment, providing an unbiased assessment of detection performance across all time steps.

### 5.1.4 Clustering Evaluation

To assess whether the learned representations preserve meaningful structure (RQ2-RQ3), clustering performance is evaluated using a variety of metrics and techniques.

#### 5.1.4.1 Adjusted Rand Index

Rand Index measures the agreement between the resulting clusters and a predefined clustering assignment. *Adjusted Rand Index* (ARI) is used in this report. ARI extends the traditional Rand Index by accounting for random cluster assignments. The score ranges from -1 to 1: a value of 0 corresponds to random assignment, and 1 indicates perfect agreement with the ground truth while a negative value means that the produced clustering is worse than what is expected by chance [65].

Formally, given a clustering result $A$ and a ground truth partition $G$, Rand Index evaluates the proportions of pairs of points in which the two partitions agree upon.

Let $n$ be the number of data points and consider all combinations of pairs. For any pair of points $(i, j)$ we define:

- $a$: number of pairs that are in the same cluster in both $A$ and $G$
- $b$: number of pairs that are in different clusters in both $A$ and $G$
- $c$: number of pairs that are in the same cluster in $A$ but different in $G$
- $d$: number of pairs that are in different clusters in $A$ but the same in $G$

The Rand Index (RI) is then defined as:

$$RI = \frac{a+b}{a+b+c+d}$$

The value of $RI$ lies in the interval $[0, 1]$, where $RI = 1$ indicates perfect agreement between the clustering result and the ground truth partition. ARI corrects this for expected agreement under random permutations.

#### 5.1.4.2 Normalised Mutual Information

Normalised Mutual Information (NMI) measures the amount of shared information between a clustering result $A$ and a ground truth partition $G$, normalised to the range $[0, 1]$:

$$\text{NMI}(A, G) = \frac{2 \cdot I(A; G)}{H(A) + H(G)},$$

where $I(A; G)$ is the mutual information between the two partitions and $H(\cdot)$ denotes entropy which measures the uncertainty in a partition. A partition with many equally sized clusters has high entropy, while one dominated by a single cluster has low entropy. Dividing by the average entropy of the two partitions normalises the score to $[0, 1]$ regardless of how many clusters each partition contains. This makes NMI less sensitive to the number of clusters than ARI, which can yield lower scores when $K$ differs substantially between the clustering result and the reference [66].

#### 5.1.4.3 Silhouette Score

*Silhouette-Score* measures cohesion versus separation by looking into how similar an object is to its own cluster.

Let $z_i$ be a latent representation belonging to cluster $A$. The average distance between $z_i$ and all other points in the same cluster is

$$a(i) = \frac{1}{|A| - 1} \sum_{z_j \in A, j \neq i} d(z_i, z_j)$$

Let $B$ denote the nearest neighbouring cluster to $A$. The average distance between $z_i$ and points in this cluster is then

$$b(i) = \frac{1}{|B|} \sum_{z_j \in B} d(z_i, z_j)$$

and the Silhouette score for point $z_i$ is defined by:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}.$$

The resulting Silhouette score ranges between $[-1, 1]$, where values close to 1 indicate well-separated clusters, values around 0 indicate overlapping clusters, and negative values indicate potential misclassification [67].

Silhouette score can be computed independently of ground truth and is used to assess cohesion within and separation between clusters. For a given clustering result, the overall score is obtained by averaging the silhouette score over all data points, providing a measure of cluster quality that can be compared across different models. The silhouette score can also be used to guide the selection of the number of clusters in algorithms that require this as input, by choosing the value that maximises the average score [68, 69].

#### 5.1.4.4 Stability Under Perturbation

ARI is also used to measure the consistency of cluster assignments under small perturbations of both the data and the model. A grouping is stable if small changes (perturbations) in the data do not cause substantial changes in the produced cluster assignments. Stable groupings are important to ensure that the discovered structures are not the result of randomness, variable initialisations, or sampling variation in the clustering algorithm [70] (RQ2).

Given the latent representations $Z$, we also generate a perturbed version $Z'$ by adding Gaussian noise to the input windows $X$ prior to encoding. The same clustering algorithm is applied to both $Z$ and $Z'$, producing assignments $c$ and $c'$ respectively. Stability is quantified as the ARI between $c$ and $c'$. High agreement indicates that the cluster structure is robust to small input perturbations rather than reflecting noise [71].

Additional perturbation strategies include bootstrapping (resampling observations) and retraining the encoder with different random initialisations. Across all strategies, stable representations should produce highly consistent cluster assignments.

### 5.1.5 Interpretability Evaluation

A clustering is considered interpretable if the resulting groups can be explained by distinct patterns in the original feature space. The resulting clusters should therefore be explainable and differentiable in what underlying feature pattern they represent. The feature pattern of one cluster should be distinct from the feature pattern of another cluster and correspond to different subsets of features (RQ3).

To characterise the underlying behaviour represented by a cluster, we compute the average feature vector for each cluster $A \subseteq Z$,

$$\bar{x}_A = \frac{1}{|A|} \sum_{i \in A} x_i.$$

The vector $\bar{x}_A$ acts as a cluster prototype in the original feature space and provides a compact summary of the typical behaviour associated with the cluster. By comparing cluster prototypes, it becomes possible to identify which features differentiate the discovered latent-space groupings.

In addition to analysing the original feature values, we also examine reconstruction-error patterns to understand how the representation learning model perceives the observations assigned to each cluster. Each window $x_i$ is associated with a feature contribution vector $a^{(i)} \in \mathbb{R}^d$, where $d$ is the number of features. Each element $a_j^{(i)}$ quantifies the contribution of feature $j$ to the representation of window $x_i$. In reconstruction-based representation learning models such as autoencoders, these contributions can be derived from feature-wise reconstruction errors [1, 44, 72]. Given an input window $x_i$ and its reconstruction $\hat{x}_i$, the overall reconstruction error is

$$s(x_i) = ||x_i - \hat{x}_i||^2$$

Adapting this, we can compute the feature-level contribution by

$$a_j^{(i)} = ||x_{ij} - \hat{x}_{ij}||^2.$$

For a cluster $A \subseteq Z$, the average reconstruction-error profile is defined by

$$\bar{a}_A = \frac{1}{|A|} \sum_{i \in A} a^{(i)}.$$

The vector $\bar{a}_A$ provides a complementary description of the cluster by highlighting which features contribute most to reconstruction errors within that group. While $\bar{x}_A$ characterises the typical behaviour represented by a cluster, $\bar{a}_A$ characterises how the learned representation models that behaviour. Together, these measures provide insight into both the structure captured by the latent space and the recurring feature patterns associated with anomalous or difficult-to-reconstruct observations [7, 38, 44, 73].

### 5.1.6 Computational Cost Metrics

To address RQ4, computational efficiency is measured along four dimensions: model size (number of trainable parameters), computational cost (floating-point operations per forward pass), training time (wall-clock time for training the neural network methods and fitting the traditional ones), and inference latency (wall-clock time per sample).

For neural network models, parameter counts are obtained by summing all learnable weights, and forward FLOPs are estimated using the `thop` library on a single input window. Total training FLOPs are approximated as $3 \times$ forward FLOPs $\times$ batches per epoch $\times$ epochs, where the factor of three accounts for the forward and backward passes. Inference latency is measured by passing a batch of 32 windows through the model, repeating five times after a warmup pass, and dividing the total elapsed time by the number of samples processed. GPU synchronisation is enforced between repetitions when applicable.

For traditional models (PCA, RP), parameters correspond to the projection matrix ($d \times k$ entries, where $d$ is the flattened window dimensionality and $k$ the number of components). Forward FLOPs are computed analytically as $2dk$ (one projection and one back-projection). Inference latency is measured identically to neural models: a warmup reconstruction call followed by five timed repetitions on a batch of 32 flattened windows. For KPCA, the kernel-based reconstruction depends on the number of training samples rather than a fixed weight matrix, so FLOPs are not reported. Instead, only inference latency and training time are measured. Training time for all models is recorded as wall-clock time from the start of the fit procedure to convergence or epoch completion.

## 5.2 Datasets

The datasets used for evaluating the approach are established benchmarks from prior research, including Server Machine Dataset (SMD), Soil Moisture Active Passive (SMAP), Secure Water Treatment (SWaT) [62, 63, 74].

### 5.2.1 SMD

The SMD dataset comprises multivariate time-series collected from 28 machines, each described by 38 different features. The training data contains only normal operational behaviour, data without anomalous behaviour, whereas the test set includes labelled anomalies specifying both time-stamp of anomaly and the affected features [38].

### 5.2.2 SMAP

The SMAP dataset, distributed by NASA [75], consists of spacecraft telemetry data where each feature corresponds to a sensor measurement. As with SMD, the training set represents normal behaviour, while anomalies only exist in the test set. However, in SMAP, the labels are limited to temporal intervals indicating when an anomaly occurs, and not the features contributing to the anomaly [74].

### 5.2.3 SWaT

The SWaT dataset, distributed by Singapore University of Technology and Design, contains data from a scaled-down industrial water treatment plant and includes data from 51 different sensors and actuators. The training data reflects normal system behaviour, while the test set contains multiple cyber-physical attacks. Anomalies are labelled only at the time-step level and not the features contributing to the anomalies [62]. A subset of the SWaT dataset is used for this project.

### 5.2.4 Case-Study

Data used in the industrial case-study is unlabelled, numerical log data of performance metrics. The dataset is highly heterogeneous, with individual test executions

varying in both length and dimensionality, and the set of active features differs between executions. Due to time constraints, no manual labelling is performed. However, *pseudo-labelling* is performed by comparing the resulting clusters to other clusters produced by internal tools that cluster the same entities (test-executions), but based on other data sources (mainly textual logs). This dataset is used to evaluate the scalability of the proposed approach to large-scale industrial multivariate time-series data. The evaluation within the case-study relies on comparing clustering asssignments against these pseudo-labels and the intepretability metrics defined in Section 5.1.5.

### 5.2.5 Multiple Datasets

The datasets used in this study, except for the industrial case-study, all provide ground truth labels of anomalous behaviour, either at the feature level (SMD) or at the temporal level (SMAP, SWaT). We ensure, by using multiple datasets, that the evaluation is robust across different domains, different scales and anomaly types, reducing the risk of overfitting or bias towards a specific data source. Moreover, they cover a range of dimensionalities and system contexts, which is suitable for our problem of investigating whether learned representations produce stable, interpretable and diagnostically useful groupings of anomalous behaviours. Including datasets of varying size and complexity is also important for answering RQ4 by assessing how the computational efficiency of the method is affected by the dataset properties. By demonstrating consistent results across diverse datasets, the approach can be generalised into other systems.

## 5.3 Evaluation Setup

The following subsections show the experimental and model setup.

### 5.3.1 Hardware and Software Configurations

All experiments, including model training and evaluation, were conducted on a single compute node equipped with an Intel Xeon Gold 6230N CPU (16 cores, 2.30 GHz), 64 GB RAM, and an NVIDIA Tesla T4 GPU with 15 GB VRAM. The system ran Ubuntu 22.04.5 LTS with CUDA 13 and cuDNN 9. The software environment included Python 3.12, PyTorch 2.12, NumPy 2.4, scikit-learn 1.8, and other standard scientific Python libraries. All dependencies can be found in the environment file in our Github repository[1].

### 5.3.2 Hyperparameter Optimization

All neural network models require the specification of several hyperparameters, including learning rate, batch size, hidden layer dimensions, sequence length, stride fraction, and encoding dimension. To ensure a fair comparison between models

[1]The GitHub repository can be found in github.com/jonathann000/jj-masterthesis

and minimize evaluation bias, each model was evaluated using the hyperparameter configuration that maximized its F1-score on the validation data.

The hyperparameter configurations were obtained through Bayesian optimization using the `Weights & Biases (wandb)` experiment tracking framework [76]. Bayesian optimization iteratively explores the hyperparameter space by constructing a probabilistic model of the objective function and selecting promising configurations for evaluation. This approach was chosen because the search space is too large to be exhaustively explored using methods such as grid search, while Bayesian optimization can identify high-performing configurations with substantially fewer evaluations [77].

## 5.4 Preprocessing

To ensure consistent evaluation across datasets with heterogeneous feature spaces and sampling characteristics, a common preprocessing pipeline was applied with dataset-specific adaptations where necessary. Since all datasets consist of multivariate time series, chronological splitting was used throughout in order to preserve temporal dependencies and avoid information leakage from future observations.

Missing numerical values were replaced with zeros using deterministic imputation. Non-numeric metadata columns, such as timestamps, row indices, and textual attack annotations, were removed prior to modelling. For datasets containing explicit anomaly labels, labels were retained separately from the feature matrices and only used during evaluation.

For all datasets, the training split was further divided chronologically into training and validation subsets. Feature standardisation was then performed using z-score normalisation with statistics estimated exclusively from the training subset. The learned transformation was subsequently applied to the test set. This ensured that no information from the test data influenced feature scaling.

For the SMD dataset, preprocessing and scaling were performed independently for each machine time series prior to aggregation, thereby preserving machine-specific operating distributions as all machines are independent. Similarly, preprocessing for the SMAP dataset was performed independently for each telemetry channel.

Features exhibiting zero variance were removed for datasets where constant sensor channels were present, such as SMAP. In the SMAP dataset, features were retained if variance was present in either the training or test split, since channels inactive during training may still become informative during anomalous behaviour.

For SWaT, the normal operating data was chronologically partitioned into training and testing segments, after which the attack sequences were appended to the test split. This ensured that the final test set contained both normal and anomalous operating conditions while preserving temporal ordering.

## 5.5 Results

Reporting anomaly detection and clustering results across three different datasets: SMD, SWaT and SMAP.

### 5.5.1 Anomaly Detection on SMD

The window-level anomaly detection results on SMD are presented in Table 5.1. OmniAnomaly achieves the highest F1-score (0.8018) and MCC (0.7842), followed closely by VAE (F1 = 0.7904) and AE (F1 = 0.7899). USAD also performs competitively, achieving an F1-score of 0.7859 and an MCC of 0.7583.

Among the classical approaches, PCA achieves the highest F1-score (0.7362), outperforming RP (0.7044). TranAD, LSTM, TransAE, and DAGMM obtain F1-scores between 0.72 and 0.75. DAGMM achieves the highest recall (0.9872), while TranAD achieves the highest precision (0.9333).

Training times vary substantially across methods. RP is the fastest method, requiring only 0.0023 seconds for training, while TransAE requires the longest training time (195.58 seconds). AE, USAD, and OmniAnomaly achieve strong performance while maintaining comparatively moderate training times.

| Model | F1 | MCC | Precision | Recall | Training Time (s) |
|---|---|---|---|---|---|
| PCA | 0.7362 | 0.7034 | 0.7191 | 0.7540 | 0.7737 |
| KPCA | 0.7052 | 0.6863 | 0.5683 | 0.9288 | 1.2490 |
| RP | 0.7044 | 0.6851 | 0.5686 | 0.9256 | 0.0023 |
| AE | 0.7899 | 0.7635 | 0.8103 | 0.7705 | 4.4512 |
| VAE | 0.7904 | 0.7627 | 0.7476 | 0.8383 | 22.5733 |
| LSTM | 0.7214 | 0.6896 | 0.6407 | 0.8252 | 86.1803 |
| TransAE | 0.7481 | 0.7177 | 0.6908 | 0.8158 | 195.5759 |
| TranAD | 0.7222 | 0.7190 | 0.9333 | 0.5890 | 20.9528 |
| USAD | 0.7859 | 0.7583 | 0.7290 | 0.8525 | 10.2101 |
| DAGMM | 0.7244 | 0.7126 | 0.5721 | 0.9872 | 22.7907 |
| OmniAnomaly | 0.8018 | 0.7842 | 0.8835 | 0.7339 | 4.6283 |

**Table 5.1:** Window-level anomaly detection results on SMD (machine-1-1).

The pointwise anomaly detection results on SMD are presented in Table 5.2. OmniAnomaly achieves the highest F1-score (0.7320) and MCC (0.7044), followed by AE (F1 = 0.6962, MCC = 0.6719) and VAE (F1 = 0.6952, MCC = 0.6777). USAD and TransAE achieve similar performance, with F1-scores of 0.6818 and 0.6857, respectively.

Among the classical methods, PCA achieves the highest F1-score (0.6794), while RP achieves a comparable F1-score of 0.6665. DAGMM obtains the lowest F1-score (0.5954) despite achieving perfect recall (1.0000), indicating a large number of false positives.

Compared to the window-level evaluation, the overall ranking of methods remains largely unchanged. OmniAnomaly remains the best-performing method, while AE, VAE, and USAD continue to achieve competitive performance. F1-scores and MCC values are generally lower than those observed in the window-level evaluation, reflecting the increased difficulty of pointwise anomaly detection.

| Model | F1 | MCC | Precision | Recall | Training Time (s) |
|---|---|---|---|---|---|
| PCA | 0.6794 | 0.6479 | 0.6011 | 0.7810 | 0.7737 |
| KPCA | 0.6665 | 0.6645 | 0.5056 | 0.9777 | 1.2490 |
| RP | 0.6665 | 0.6645 | 0.5056 | 0.9777 | 0.0023 |
| AE | 0.6962 | 0.6719 | 0.5902 | 0.8486 | 4.4512 |
| VAE | 0.6952 | 0.6777 | 0.5672 | 0.8979 | 22.5733 |
| LSTM | 0.6752 | 0.6534 | 0.5531 | 0.8664 | 86.1803 |
| TransAE | 0.6857 | 0.6662 | 0.5604 | 0.8831 | 195.5759 |
| TranAD | 0.6731 | 0.6399 | 0.6032 | 0.7613 | 20.9528 |
| USAD | 0.6818 | 0.6655 | 0.5476 | 0.9031 | 10.2101 |
| DAGMM | 0.5954 | 0.6030 | 0.4239 | 1.0000 | 22.7907 |
| OmniAnomaly | 0.7320 | 0.7044 | 0.6765 | 0.7973 | 4.6283 |

**Table 5.2:** Pointwise anomaly detection results on SMD (machine-1-1).

### 5.5.2 Anomaly Detection on SWaT

The window-level anomaly detection results on the SWaT dataset are presented in Table 5.3. LSTM achieves the highest F1-score (0.8882) and MCC (0.8758), followed by OmniAnomaly (F1 = 0.8792, MCC = 0.8674). Both methods achieve the highest recall values among all evaluated approaches.

Among the remaining deep learning methods, AE, VAE, and TranAD achieve F1-scores of 0.8486, 0.8333, and 0.8432, respectively, while TransAE obtains an F1-score of 0.8118. USAD and DAGMM show similar performance, with F1-scores of 0.8203 and 0.8189.

Among classical methods, RP achieves an F1-score of 0.8083, PCA achieves 0.7815, and KPCA achieves 0.2189. KPCA also obtains the lowest MCC (0.3241) among all methods.

Precision is close to 1.0 for most methods. Recall ranges from 0.6413 (PCA) to 0.8028 (LSTM), with the exception of the outlier KPCA (0.1229).

Training times differ significantly across methods, with RP requiring the shortest time (0.0028 s). LSTM, TranAD, and TransAE require the longest training times, exceeding 1000 seconds, while AE, VAE, USAD, and DAGMM fall in an intermediate range.

The pointwise anomaly detection results are presented in Table 5.4. The ranking of methods remains largely consistent with the window-level evaluation. The LSTM

model achieves the highest F1 (0.8924) and MCC-score (0.8804) followed by OmniAnomaly (F1 = 0.8800), MCC = 0.8683). TranAD also achieve strong performance, with an F1-score of 0.8444.

Among classical methods, RP achieves the highest F1-score (0.8087), followed by PCA (0.7808), while KPCA again achieves the lowest performance with an F1-score of 0.2189 and MCC of 0.3241.

Precision remains high across all methods, while recall values again vary substantially, with KPCA obtaining the lowest recall (0.1229) and LSTM achieving one of the highest values (0.8086).

Differences between window-level and pointwise results are small, and the relative ranking of methods is largely unchanged across both evaluation settings.

| Model | F1 | MCC | Precision | Recall | Training Time (s) |
|---|---|---|---|---|---|
| PCA | 0.7815 | 0.7740 | 1.0000 | 0.6413 | 0.3703 |
| KPCA [1] | 0.2189 | 0.3241 | 1.0000 | 0.1229 | 131.2726 |
| RP | 0.8083 | 0.7987 | 1.0000 | 0.6783 | 0.0028 |
| AE | 0.8486 | 0.8370 | 0.9993 | 0.7375 | 174.0305 |
| VAE | 0.8333 | 0.8223 | 1.0000 | 0.7142 | 220.0860 |
| LSTM | 0.8882 | 0.8758 | 0.9939 | 0.8028 | 2463.4042 |
| TransAE | 0.8118 | 0.8019 | 1.0000 | 0.6832 | 1486.8429 |
| TranAD | 0.8432 | 0.8316 | 0.9990 | 0.7294 | 2152.0512 |
| USAD | 0.8203 | 0.8099 | 1.0000 | 0.6953 | 165.0091 |
| DAGMM | 0.8189 | 0.8086 | 1.0000 | 0.6933 | 211.0226 |
| OmniAnomaly | 0.8792 | 0.8674 | 0.9998 | 0.7845 | 235.6670 |

**Table 5.3:** Window-level anomaly detection results on SWaT

| Model | F1 | MCC | Precision | Recall | Training Time (s) |
|---|---|---|---|---|---|
| PCA | 0.7808 | 0.7734 | 1.0000 | 0.6404 | 0.3703 |
| KPCA [1] | 0.2189 | 0.3241 | 1.0000 | 0.1229 | 131.2726 |
| RP | 0.8087 | 0.7991 | 1.0000 | 0.6789 | 0.0028 |
| AE | 0.8515 | 0.8400 | 1.0000 | 0.7415 | 174.0305 |
| VAE | 0.8354 | 0.8243 | 1.0000 | 0.7173 | 220.0860 |
| LSTM | 0.8924 | 0.8804 | 0.9957 | 0.8086 | 2463.4042 |
| TransAE | 0.8125 | 0.8026 | 1.0000 | 0.6842 | 1486.8429 |
| TranAD | 0.8444 | 0.8330 | 0.9998 | 0.7308 | 2152.0512 |
| USAD | 0.8191 | 0.8088 | 1.0000 | 0.6937 | 165.0091 |
| DAGMM | 0.8180 | 0.8078 | 1.0000 | 0.6920 | 211.0226 |
| OmniAnomaly | 0.8800 | 0.8683 | 1.0000 | 0.7858 | 235.6670 |

**Table 5.4:** Pointwise anomaly detection results on SWaT

[1]KPCA was trained on a subsampled subset of the SWaT dataset due to the computational complexity of kernel matrix computation and eigendecomposition. The aggressive subsampling leads to the poor performance.

### 5.5.3 Anomaly Detection on SMAP

The window-level anomaly detection results on the SMAP dataset are presented in Table 5.5. OmniAnomaly achieves the highest overall performance, obtaining an F1-score of 0.9431 and an MCC of 0.9145. USAD achieves the second-highest F1-score (0.9393), followed by LSTM (0.9349), TranAD (0.9275), and AE (0.9273).

Among the classical dimensionality reduction methods, PCA achieves the strongest performance with an F1-score of 0.9208 and MCC of 0.8726. KPCA performs similarly, obtaining an F1-score of 0.9145. In contrast, RP achieves substantially lower performance, with an F1-score of 0.5689 and MCC of 0.4896.

Precision values exceed 0.89 for all methods and reach 1.0 for OmniAnomaly. Differences in performance are therefore primarily driven by recall. PCA, KPCA, and TransAE achieve the highest recall values, all exceeding 0.90, whereas RP achieves the lowest recall (0.4173). Training times vary considerably across methods, ranging from 0.0012 seconds for RP to approximately 22 seconds for LSTM and TransAE.

| Model | F1 | MCC | Precision | Recall | Training Time (s) |
|---|---|---|---|---|---|
| PCA | 0.9208 | 0.8726 | 0.9338 | 0.9081 | 0.0924 |
| KPCA | 0.9145 | 0.8622 | 0.9248 | 0.9044 | 0.7595 |
| RP | 0.5689 | 0.4896 | 0.8937 | 0.4173 | 0.0012 |
| AE | 0.9273 | 0.8847 | 0.9534 | 0.9026 | 1.5266 |
| VAE | 0.9126 | 0.8663 | 0.9671 | 0.8640 | 1.9661 |
| LSTM | 0.9349 | 0.8995 | 0.9785 | 0.8950 | 22.0274 |
| TransAE | 0.9113 | 0.8567 | 0.9149 | 0.9078 | 22.0925 |
| TranAD | 0.9275 | 0.8867 | 0.9636 | 0.8940 | 2.9171 |
| USAD | 0.9393 | 0.9077 | 0.9915 | 0.8923 | 0.5988 |
| DAGMM | 0.8723 | 0.8057 | 0.9367 | 0.8162 | 2.7653 |
| OmniAnomaly | 0.9431 | 0.9145 | 1.0000 | 0.8923 | 0.7452 |

**Table 5.5:** Window-level anomaly detection on SMAP (D-1)

The pointwise results shown in Table 5.6 closely mirror the window-level results. USAD achieves the highest F1-score (0.9417), followed by OmniAnomaly (0.9374) and VAE (0.9355). PCA and KPCA remain the strongest classical methods, while RP continues to exhibit substantially lower performance. For all methods, the differences in F1-score and MCC between the two evaluation protocols are below 0.03, indicating that the conclusions are largely independent of the evaluation granularity

### 5.5.4 Stability on SMD

Table 5.7 shows clustering stability results under induced Gaussian noise ($\sigma = 2.0$) together with the corresponding ARI and Silhouette scores. To facilitate fair comparison across noise levels, clustering hyperparameters were fixed throughout the experiments. K-Means and FCM were evaluated using $K = 4$, while DBSCAN was evaluated using $min\_samples = 10$ and $\varepsilon = 1.0$. Although DBSCAN is a density-based method and therefore not directly comparable to the centroid-based

| Model | F1 | MCC | Precision | Recall | Training Time (s) |
|---|---|---|---|---|---|
| PCA | 0.9225 | 0.8767 | 0.9435 | 0.9024 | 0.0573 |
| KPCA | 0.9208 | 0.8728 | 0.9326 | 0.9092 | 0.8114 |
| RP | 0.5871 | 0.5199 | 0.9272 | 0.4296 | 0.0047 |
| AE | 0.9298 | 0.8904 | 0.9659 | 0.8963 | 1.6701 |
| VAE | 0.9355 | 0.9011 | 0.9824 | 0.8929 | 2.2600 |
| LSTM | 0.9328 | 0.8956 | 0.9727 | 0.8960 | 21.9978 |
| TransAE | 0.9238 | 0.8780 | 0.9383 | 0.9098 | 22.4339 |
| TranAD | 0.9252 | 0.8818 | 0.9528 | 0.8990 | 2.9671 |
| USAD | 0.9417 | 0.9126 | 1.0000 | 0.8898 | 0.6070 |
| DAGMM | 0.8697 | 0.8058 | 0.9485 | 0.8030 | 2.8918 |
| OmniAnomaly | 0.9374 | 0.9067 | 1.0000 | 0.8822 | 0.8869 |

**Table 5.6:** Pointwise anomaly detection results on SMAP (D-1)

approaches, it was included to provide a contrasting clustering paradigm and to investigate whether clustering stability under noise differs between density-based and centroid-based methods.

The results show that RP, AE, VAE, USAD, and OmniAnomaly achieve perfect stability with an ARI of 1.0, while TranAD and PCA exhibit high stability with ARI values of 0.978 and 0.853, respectively. KPCA, LSTM, DAGMM, and TransAE show lower stability with ARI ranging from 0.165 to 0.390. OmniAnomaly's perfect ARI is obtained by DBSCAN, for which the Silhouette score is undefined (nan) due to all points being assigned to a single cluster.

The difference between the Silhouette score before and after inducing noise varies between models but mostly remains stable. AE, VAE, RP, USAD and TranAD show minimal difference ($\leq 0.009$). PCA exhibits a larger drop (0.260), followed by KPCA (0.288), LSTM (0.267), TransAE (0.321), and DAGMM (0.240). The result for OmniAnomaly is not present under these configurations, but the full table with results across all noise levels tested ($\sigma \in 0.1, 0.5, 1.0, 2.0$) and clustering algorithms are reported in Appendix 1 (Table 8.2)

Tables 5.8–5.9 show results on a combined version of the SMD dataset, consisting of six different machines with distinct behaviours. From these machines, a subset of datapoints that are labelled as anomalies was applied to each encoder, the resulting latent space was then used as input to clustering algorithms, and the original machine identity is then used as the ground truth label for measuring cluster performance.

The clustering performance across different representation spaces is shown in Table 5.8. The results show that clustering performance, measured using Adjusted Rand Index (ARI), varies depending on both the encoder and the representation space. Overall, the LSTM-based encoder achieves the highest ARI (0.614) when

[1]Silhouette score before inducing noise
[2]Silhouette score after inducing noise

| Model | Best Clustering | Stability (ARI) | ARI std | $\text{Sil}_a$[1] | $\text{Sil}_b$[2] |
|---|---|---|---|---|---|
| RP | KMeans | 1.000 | 0.000 | 0.998 | 0.995 |
| AE | KMeans | 1.000 | 0.000 | 0.995 | 0.994 |
| VAE | KMeans | 1.000 | 0.000 | 0.998 | 0.995 |
| USAD | KMeans | 1.000 | 0.000 | 0.977 | 0.968 |
| OmniAnomaly | DBSCAN | 1.000 | 0.000 | nan | nan |
| TranAD | FCM | 0.978 | 0.022 | 0.998 | 0.996 |
| PCA | KMeans | 0.853 | 0.010 | 0.631 | 0.371 |
| KPCA | KMeans | 0.390 | 0.021 | 0.378 | 0.090 |
| LSTM-AE | KMeans | 0.322 | 0.010 | 0.459 | 0.192 |
| DAGMM | FCM | 0.170 | 0.005 | 0.561 | 0.321 |
| Trans-AE | KMeans | 0.165 | 0.008 | 0.489 | 0.168 |

**Table 5.7:** Clustering stability at $\sigma = 2.0$ on SMD machine-1-1, showing the best-performing clustering algorithm per encoder

combined with UMAP, outperforming all other evaluated configurations. PCA combined with UMAP also yields strong performance (0.572), while clustering directly on raw features results in a lower ARI of 0.237. Across encoders, performance generally improves when applying non-linear dimensionality reduction techniques such as UMAP compared to clustering in the original latent space.

Table 5.9 presents the best-performing clustering configuration for each encoder, selected based on the highest ARI score across evaluated clustering setups. In most cases, clustering in the UMAP-transformed space yields higher ARI and NMI compared to clustering directly in the latent space. The LSTM-based encoder combined with KMeans clustering achieves the highest overall performance in terms of ARI (0.614) and NMI (0.725), while also showing a competitive silhouette score (0.590). TransAE produces the highest silhouette score (0.678), indicating strong cluster separation according to this metric, although with a lower ARI compared to the best-performing LSTM configuration. The reported results represent the best observed configuration per encoder across the tested clustering and representation combinations, as selected by ARI. Full results are available in Table 8.4 in the Appendix.

### 5.5.5 Stability of SWaT

Table 5.10 summarises clustering stability results under induced Gaussian noise ($\sigma = 2.0$) together with the corresponding ARI and Silhouette scores. To facilitate comparison across noise levels, clustering hyperparameters were fixed throughout the experiments. K-Means and FCM were evaluated using K=6, while DBSCAN was evaluated using $min\_samples = 10$ and $\varepsilon = 1.5$. DBSCAN is, as in Stability on SMD (See 5.5.4 above), included to give a contrast between centroid-based and density-based clustering methods.

The results show that VAE, RP, and PCA have perfect stability with an ARI of 1.0, while KPCA, TranAD, AE, USAD, LSTM, DAGMM, and TransAE have high

| Encoder | Raw | Latent | t-SNE | UMAP |
|---|---|---|---|---|
| Raw | 0.237 | – | – | – |
| PCA | – | 0.486 | 0.465 | 0.572 |
| KPCA | – | 0.075 | 0.310 | 0.335 |
| RP | – | 0.164 | 0.204 | 0.248 |
| AE | – | 0.418 | 0.499 | 0.500 |
| VAE | – | 0.359 | 0.412 | 0.424 |
| LSTM | – | 0.349 | 0.459 | 0.614 |
| TransAE | – | 0.325 | 0.519 | 0.491 |
| USAD | – | 0.225 | 0.347 | 0.309 |
| DAGMM | – | 0.306 | 0.325 | 0.342 |
| OmniAnomaly | – | 0.298 | 0.239 | 0.199 |

**Table 5.8:** Clustering performance (ARI) across representation spaces (SMD).

| Encoder | Space | Metric | Clustering | ARI | NMI | Silhouette |
|---|---|---|---|---|---|---|
| Raw | Raw | cosine | KMeans | 0.237 | 0.429 | 0.102 |
| PCA | Latent+UMAP | euclidean | KMeans | 0.572 | 0.679 | 0.582 |
| KPCA | Latent+UMAP | euclidean | FCM | 0.335 | 0.387 | 0.611 |
| RP | Latent+UMAP | cosine | KMeans | 0.248 | 0.311 | 0.504 |
| AE | Latent+UMAP | euclidean | DBSCAN | 0.500 | 0.625 | 0.547 |
| VAE | Latent+UMAP | euclidean | KMeans | 0.424 | 0.506 | 0.458 |
| LSTM | Latent+UMAP | euclidean | KMeans | 0.614 | 0.725 | 0.590 |
| TransAE | Latent+UMAP | euclidean | DBSCAN | 0.491 | 0.701 | 0.678 |
| USAD | Latent+UMAP | euclidean | DBSCAN | 0.309 | 0.514 | 0.301 |
| DAGMM | Latent+UMAP | euclidean | KMeans | 0.342 | 0.400 | 0.467 |
| OmniAnomaly | Latent | cosine | KMeans | 0.298 | 0.388 | 0.130 |

**Table 5.9:** Best clustering configuration per encoder selected by highest ARI (SMD).

stability with ARI values ranging from $0.930 - 0.996$. OmniAnomaly exhibits the lowest stability at ARI $= 0.521$.

Regarding cluster quality, the Silhouette score difference between before and after inducing noise varies across different models. VAE shows zero change while AE and RP show minimal differences ($\leq 0.009$). KPCA, DAGMM, and TransAE show larger drops in Silhouette score with a difference ranging from 0.060 to 0.153, and OmniAnomaly shows the largest degradation (0.255). The full table with results across all noise levels tested ($\sigma \in \{0.1, 0.5, 1.0, 2.0\}$ and clustering algorithms is reported in Appendix 1 (Table 8.1).

[1]Silhouette score before inducing noise
[2]Silhouette score after inducing noise

| Model | Best Clustering | Stability (ARI) | ARI std | $Sil_a$[1] | $Sil_b$[2] |
|---|---|---|---|---|---|
| VAE | KMeans | 1.000 | 0.000 | 0.962 | 0.962 |
| RP | DBSCAN | 1.000 | 0.000 | 0.955 | 0.946 |
| PCA | FCM | 1.000 | 0.000 | 0.580 | 0.570 |
| KPCA | KMeans | 0.996 | 0.002 | 0.904 | 0.850 |
| TranAD | FCM | 0.995 | 0.002 | 0.613 | 0.566 |
| AE | DBSCAN | 0.995 | 0.006 | 0.945 | 0.943 |
| USAD | FCM | 0.978 | 0.005 | 0.736 | 0.714 |
| LSTM | KMeans | 0.973 | 0.019 | 0.549 | 0.498 |
| DAGMM | KMeans | 0.930 | 0.010 | 0.639 | 0.486 |
| TransAE | KMeans | 0.930 | 0.023 | 0.605 | 0.545 |
| OmniAnomaly | FCM | 0.521 | 0.012 | 0.440 | 0.185 |

**Table 5.10:** Clustering stability at $\sigma = 2.0$ on SWaT, showing the best-performing clustering algorithm per autoencoder architecture

### 5.5.6 Stability on SMAP

Table 5.11 shows the best stability results under Gaussian noise ($\sigma = 2.0$) for the SMAP (D-1) dataset. The hyperparameters in this dataset were fixed for fuzzy C-Means and K-Means with $K = 3$ and DBSCAN with $min\_samples = 10$ and $\varepsilon = 0.8$.

None of the combinations achieved perfect stability with TranAD achieving the highest with an ARI of 0.858, while most of the models, KPCA, PCA, LSTM, and TransAE, exhibit moderate stability between $0.673 - 0.814$. OmniAnomaly, VAE, RP, USAD, and DAGMM show lower stability with ARI values ranging from 0.411 to 0.650.

Regarding cluster quality, LSTM shows the smallest Silhouette score degradation (0.032), while DAGMM, VAE, and RP show the largest (0.278-0.341). Notably, TranAD achieves the highest ARI stability but suffers a substantial drop in cluster separability (0.256). The full table with results across all noise tested ($\sigma \in 0.1, 0.5, 1.0, 2.0$) and clustering algorithms are reported in Appendix 1 (Table 8.3)

The clustering results are presented in Table 5.12 and Table 5.13. In the raw feature space, the baseline achieves an ARI of 0.429. Applying t-SNE and UMAP to the raw features results in slightly lower ARI values of 0.378 and 0.393, respectively.

In the latent feature space, performance varies across encoders and embedding methods. The highest ARI in this space is obtained by OmniAnomaly with 0.490 in the latent representation, followed by TransAE (0.414) and KPCA (0.407). PCA achieves an ARI of 0.331, while RP obtains 0.378. AE, VAE, and DAGMM achieve intermediate values between 0.385 and 0.397. LSTM shows its highest ARI of 0.438

[1]Silhouette score before inducing noise
[2]Silhouette score after inducing noise

| Model | Best Clustering | Stability (ARI) | ARI std | $\text{Sil}_a$[1] | $\text{Sil}_b$[2] |
|---|---|---|---|---|---|
| TranAD | KMeans | 0.858 | 0.003 | 0.846 | 0.590 |
| KPCA | KMeans | 0.814 | 0.017 | 0.558 | 0.451 |
| PCA | FCM | 0.767 | 0.024 | 0.533 | 0.374 |
| LSTM | FCM | 0.737 | 0.014 | 0.590 | 0.558 |
| AE | KMeans | 0.727 | 0.023 | 0.498 | 0.265 |
| TransAE | FCM | 0.673 | 0.020 | 0.588 | 0.484 |
| OmniAnomaly | FCM | 0.650 | 0.013 | 0.356 | 0.242 |
| VAE | FCM | 0.611 | 0.015 | 0.559 | 0.224 |
| RP | FCM | 0.554 | 0.008 | 0.435 | 0.157 |
| USAD | KMeans | 0.540 | 0.042 | 0.627 | 0.458 |
| DAGMM | FCM | 0.411 | 0.011 | 0.543 | 0.202 |

**Table 5.11:** Clustering stability at $\sigma = 2.0$ on SMAP D-1, showing the best-performing clustering algorithm per encoder

when combined with UMAP.

Across embedding methods, t-SNE and UMAP do not consistently improve ARI relative to the latent representations. For several encoders, including VAE and RP, latent-space clustering yields comparable or higher ARI than the corresponding embedded variants.

Table 5.13 reports the best-performing clustering configuration per encoder in terms of ARI, along with the corresponding NMI and silhouette score. The highest ARI is obtained by OmniAnomaly (0.490), followed by LSTM (0.438) and TransAE (0.430). PCA-based methods achieve their best performance in t-SNE space (ARI = 0.397), while KPCA performs best in latent space (ARI = 0.407). The highest silhouette score is observed for KPCA (0.651), while the highest NMI is obtained by OmniAnomaly (0.561).

### 5.5.7 Interpretability

Building on the stability results from Section 5.5.4, where the LSTM encoder combined with UMAP projection and K-Means clustering (Euclidean distance) achieved the highest clustering stability, we evaluate whether the resulting groupings are also interpretable. Interpretability is assessed on the same subset of six SMD machines used in the stability experiment (machines 1-1, 1-7, 2-2, 2-4, 3-6, and 3-8), by examining whether anomaly patterns within each cluster are distinct and can be traced back to the underlying system, in this case, the machine from which they originate.

We evaluate interpretability through two complementary measures: the cluster prototype $\bar{x}_A$, which summarises the typical feature-level behaviour of each group, and the reconstruction-error profile $\bar{a}_A$, which characterises how the learned representation perceives each group's anomalies. For each measure, we compute pairwise cosine distances between all machine pairs to quantify distinctness. A high cosine

| Raw Feature Space | | |
|---|---|---|
| Raw | Raw+tSNE | Raw+UMAP |
| 0.429 | 0.378 | 0.393 |

| Latent Feature Space | | | |
|---|---|---|---|
| Encoder | Latent | Latent+tSNE | Latent+UMAP |
| PCA | 0.331 | 0.397 | 0.302 |
| KPCA | 0.407 | 0.364 | 0.250 |
| RP | 0.378 | 0.382 | 0.315 |
| AE | 0.397 | 0.304 | 0.361 |
| VAE | 0.392 | 0.291 | 0.291 |
| LSTM | 0.372 | 0.335 | 0.438 |
| TransAE | 0.414 | 0.360 | 0.430 |
| USAD | 0.379 | 0.310 | 0.402 |
| DAGMM | 0.385 | 0.407 | 0.357 |
| OmniAnomaly | 0.490 | 0.418 | 0.360 |

**Table 5.12:** Clustering performance (ARI) in raw and latent feature spaces with different embeddings (SMAP).

| Encoder | Space | Metric | Clustering | ARI | NMI | Silhouette |
|---|---|---|---|---|---|---|
| Raw | Raw | cosine | KMeans | 0.429 | 0.499 | 0.326 |
| PCA | Latent+tSNE | euclidean | FCM | 0.397 | 0.458 | 0.424 |
| KPCA | Latent | euclidean | FCM | 0.407 | 0.480 | 0.651 |
| RP | Latent+tSNE | euclidean | FCM | 0.382 | 0.422 | 0.416 |
| AE | Latent+UMAP | cosine | KMeans | 0.361 | 0.425 | 0.654 |
| VAE | Latent | cosine | FCM | 0.392 | 0.435 | 0.457 |
| LSTM | Latent+UMAP | euclidean | DBSCAN | 0.438 | 0.530 | 0.429 |
| TransAE | Latent+UMAP | euclidean | FCM | 0.430 | 0.497 | 0.584 |
| USAD | Latent+UMAP | cosine | KMeans | 0.402 | 0.461 | 0.691 |
| DAGMM | Latent+tSNE | cosine | FCM | 0.407 | 0.461 | 0.507 |
| OmniAnomaly | Latent | euclidean | KMeans | 0.490 | 0.561 | 0.380 |

**Table 5.13:** Best clustering configuration per encoder selected by highest ARI (SMAP).

distance indicates that the clusters are dominated by different feature subsets and are therefore explainable.

#### 5.5.7.1 Cluster Prototypes

The cluster prototypes $\bar{x}_A$, seen in Table 5.14, reveal the average feature values associated with each machine's anomalous windows. The mean pairwise cosine distance between prototypes is 0.859, indicating that the clusters occupy largely orthogonal directions in feature space. Table 5.14 shows the top-3 dominant features per ma-

chine. Machine-2-4 is the most isolated cluster, with cosine distances exceeding 1.0 to three other machines, driven by feature F22 which appears in no other machine's top features. Machines 2-2 and 3-8 exhibit the lowest pairwise distance (0.339), sharing features F3 and F6, suggesting partial overlap in operating behaviour despite belonging to different server groups. The remaining clusters are well-separated, each dominated by distinct feature subsets.

#### 5.5.7.2 Reconstruction-Error Profiles

The reconstruction-error profiles $\bar{a}_A$, in Table 5.14, show which features the model finds difficult to reconstruct for each cluster's anomalies. The mean pairwise cosine distance between error profiles is 0.791. While lower than the prototype distinctness, this confirms that the model associates different failure modes with different features. However, machines 1-7 and 3-6 share nearly identical error profiles (cosine distance 0.012), both dominated by feature F9. This indicates that the LSTM struggles to reconstruct F9 across multiple machines regardless of their group membership. In contrast, machines 1-1, 2-2, and 2-4 each exhibit highly distinctive error profiles (pairwise distances $> 0.97$), with no overlap in their top contributing features.

**Table 5.14:** Interpretability results for the LSTM encoder on SMD. Top-3 dominant features are shown for both the cluster prototype $\bar{x}_A$ and the reconstruction-error profile $\bar{a}_A$.

| Machine | Prototype $\bar{x}_A$ top-3 | Error profile $\bar{a}_A$ top-3 | Overlap |
|---|---|---|---|
| machine-1-1 | F33, F32, F29 | F33, F32, F9 | 2/3 |
| machine-1-7 | F32, F33, F9 | F9, F10, F32 | 2/3 |
| machine-2-2 | F3, F2, F1 | F2, F1, F3 | 3/3 |
| machine-2-4 | F22, F6, F5 | F22, F19, F13 | 1/3 |
| machine-3-6 | F9, F6, F12 | F9, F6, F32 | 2/3 |
| machine-3-8 | F6, F3, F5 | F9, F6, F3 | 1/3 |

| Metric | Prototype $\bar{x}_A$ | Error profile $\bar{a}_A$ |
|---|---|---|
| Mean pairwise cosine distance | 0.859 | 0.791 |
| Cluster validity (ARI / NMI) | 0.611 / 0.711 | |
| Intra/inter distance ratio | 0.680 | |

### 5.5.8 Case Study – Anomaly Pattern

Using full granularity on each test run has shown to be a difficult task. The different test runs differ greatly in size (even when each failure window only contains 10 minutes worth of data), with some test executions having $> 100\,000$ features, while others having $< 20\,000$ features. This is problematic as models using cross-feature mechanisms such as LSTM need all input data to have the same dimensionality. Due to size, PCA and KPCA are not feasible, with covariance matrices having $O(d^2)$ space complexity, meaning that for the largest test executions, the covariance matrix alone would require $> 90$ GB of memory (for 8-byte floats).

Random Projection achieves a matching accuracy of 2/3, correctly identifying two

of three held-out runs to their respective pseudo-label cluster. The LSTM autoencoder with UMAP dimensionality reduction to 3D and KMeans clustering ($k = 3$, Euclidean distance) achieves an Adjusted Rand Index of 0.28 against the pseudo-labels and a Silhouette score of 0.51, with a matching accuracy of 2/3.

The strongest result is observed for runs of the same test type, which achieve a cosine similarity of 0.99 in the mean latent space. For runs of related but not identical test types within the same pseudo-label cluster, the pipeline achieves a cosine similarity of 0.88, correctly identifying them as belonging to the same group. This demonstrates that the encoder produces near-identical fingerprints for repeated executions of the same test, and similar fingerprints for related test variants sharing the same pseudo-label.

#### 5.5.8.1 Cluster Prototypes

The cluster prototypes $\bar{x}_A$ reveal the average feature values associated with each discovered cluster. As summarised in Table 5.15, the K-Means algorithm identifies three groups that do not fully align with the pseudo-labels. Due to confidentiality constraints, the numerical prototype values cannot be reported. Instead, Table 5.15 summarises the cluster composition and the dominant feature groups used in the analysis.

**Table 5.15:** Summary of the discovered K-Means clusters.

| Cluster | Windows | Pseudo-labels | Dominant feature groups |
|---|---|---|---|
| 0 | 186 | B and C | Disk synchronisation, network throughput, memory allocation. Features originate from the same system component. |
| 1 | 148 | B and A | Memory utilisation, timestamp-related counters. |
| 2 | 108 | C and A | Persistent volume usage, database-related metrics. |

As shown in Table 5.15, each cluster is characterised by a distinct subset of dominant features. Cluster 0 is exclusively associated with distributed coordination service metrics, whereas Cluster 1 is characterised by memory utilisation and timestamp-related metrics. Cluster 2 is dominated by persistent volume usage and database-related metrics. The non-overlap in dominant feature groups across the clusters suggests that the latent space captures structurally different system behaviours.

#### 5.5.8.2 Reconstruction-Error Profiles

The reconstruction-error profiles $\bar{a}_A$ show which features the model finds difficult to reconstruct for each cluster. The three clusters exhibit clearly different error magnitudes and feature compositions:

- Cluster 0 has the highest maximum per-feature error (36.0), dominated by distributed coordination and object storage metrics. The top 7 features all belong to the same component.

- Cluster 1 has a maximum per-feature error of 39.1, dominated by memory allocation metrics and storage throughput metrics across multiple components.

- Cluster 2 has the lowest error magnitudes (maximum 16.2), dominated by CPU utilisation and database duration metrics.

The separation between error profiles confirms that the model associates different system behaviours with different latent regions. Notably, the feature categories that dominate $\bar{a}_A$ largely overlap with those dominating $\bar{x}_A$ for the same cluster, indicating consistency between what the system was doing and what the model found difficult to reconstruct.

The fact that the discovered clusters cut across pseudo-label boundaries, with the model grouping runs by dynamics similarity rather than error-message similarity, is discussed further in Section 5.7.

## 5.6 Memory and Computational Cost

Tables 5.16–5.18 compare anomaly detection performance against computational cost across the evaluated datasets. The reported metrics include F1-score, number of trainable parameters, floating-point operations (FLOPs), training time, and inference time.

On SMD (Table 5.16), OmniAnomaly achieves the highest F1-score (0.8018), followed by VAE (0.7904), AE (0.7899), and USAD (0.7859). The LSTM model has the largest parameter count (1,056,886) and highest FLOP count (39.3M), while PCA and RP require substantially fewer parameters and computational operations. RP exhibits the lowest training time (0.003 s) and among the lowest inference times (0.0038 ms), whereas TransAE records the longest training time (195.58 s).

On SWaT (Table 5.17), the LSTM model achieves the highest F1-score (0.8882), followed by OmniAnomaly (0.8783). PCA attains an F1-score of 0.8212 while requiring only 20,400 parameters and 40,800 FLOPs. RP again exhibits the lowest training and inference times among the evaluated methods. LSTM and TranAD incur the highest training costs, requiring 2463.40 s and 2152.05 s respectively.

On SMAP (Table 5.18), OmniAnomaly achieves the highest F1-score (0.9431), closely followed by USAD (0.9393) and LSTM (0.9349). PCA attains an F1-score of 0.9208 with 9,600 parameters and 19,200 FLOPs. RP remains the least computationally demanding method in terms of training and inference time but records a substantially lower F1-score (0.5689) compared to the remaining methods.

Across all datasets, PCA and RP consistently require the fewest parameters, FLOPs, and shortest execution times, while LSTM-based models generally exhibit the highest parameter counts and computational requirements. The highest F1-scores are achieved by different models depending on the dataset, with OmniAnomaly obtaining the best results on SMD and SMAP, and LSTM obtaining the best result on SWaT.

| Model | F1 | Params | FLOPs | Train (s) | Infer (ms) |
|---|---|---|---|---|---|
| OmniAnomaly | 0.8018 | 63,050 | 3,193,600 | 4.63 | 0.0260 |
| VAE | 0.7904 | 508,812 | 507,136 | 22.57 | 0.0183 |
| AE | 0.7899 | 507,772 | 506,112 | 4.45 | 0.0175 |
| USAD | 0.7859 | 190,240 | 310,400 | 10.21 | 0.0342 |
| TranAD | 0.7481 | 163,262 | 1,748,000 | 20.35 | 0.6944 |
| TransAE | 0.7481 | 197,622 | 3,650,048 | 195.58 | 0.1378 |
| PCA | 0.7362 | 15,200 | 30,400 | 1.04 | 0.0054 |
| DAGMM | 0.7244 | 189,248 | 190,979 | 22.79 | 0.2606 |
| LSTM | 0.7226 | 1,056,886 | 39,346,752 | 76.88 | 0.3828 |
| KPCA | 0.7052 | – | – | 1.249 | 0.0160 |
| RP | 0.7044 | 15,200 | 30,400 | 0.003 | 0.0038 |

**Table 5.16:** Cost-performance comparison on SMD.

| Model | F1 | Params | FLOPs | Train (s) | Infer (ms) |
|---|---|---|---|---|---|
| LSTM | 0.8882 | 1,071,475 | 20,053,952 | 2463.40 | 0.3606 |
| OmniAnomaly | 0.8783 | 66,391 | 3,360,000 | 245.17 | 0.0354 |
| TranAD | 0.8432 | 284,045 | 495,720 | 2152.05 | 0.5438 |
| AE | 0.8424 | 674,822 | 672,512 | 176.82 | 0.0172 |
| VAE | 0.8388 | 675,862 | 673,536 | 226.30 | 0.0196 |
| DAGMM | 0.8216 | 252,948 | 255,329 | 219.65 | 0.0236 |
| PCA | 0.8212 | 20,400 | 40,800 | 5.47 | 0.0077 |
| USAD | 0.8202 | 253,940 | 414,400 | 181.59 | 0.0202 |
| TransAE | 0.8118 | 353,467 | 2,030,848 | 1486.84 | 0.0687 |
| RP | 0.8028 | 20,400 | 40,800 | 0.008 | 0.0045 |
| KPCA | 0.2909 | – | – | 682.13 | 0.0996 |

**Table 5.17:** Cost-performance comparison on SWaT.

## 5.7 Discussion

This section discusses the results in depth, analysing why certain methods perform as observed and identifying implications for the research questions.

### 5.7.1 Anomaly Detection

The result suggests that no model consistently outperforms across datasets. Instead, the performance of the different methods depends on characteristics of the datasets and granularity of evaluation (window-level vs pointwise evaluation).

| Model | F1 | Params | FLOPs | Train (s) | Infer (ms) |
|---|---|---|---|---|---|
| OmniAnomaly | 0.9431 | 59,452 | 3,014,400 | 0.75 | 0.0264 |
| USAD | 0.9393 | 121,640 | 198,400 | 0.60 | 0.0306 |
| LSTM | 0.9349 | 1,040,744 | 38,540,352 | 22.03 | 0.1580 |
| TranAD | 0.9275 | 70,232 | 1,036,800 | 2.92 | 0.1615 |
| AE | 0.9273 | 327,872 | 326,912 | 1.53 | 0.0192 |
| PCA | 0.9208 | 9,600 | 19,200 | 0.09 | 0.0041 |
| KPCA | 0.9145 | – | – | 0.76 | 0.0726 |
| VAE | 0.9126 | 328,912 | 327,936 | 1.97 | 0.0175 |
| TransAE | 0.9113 | 195,816 | 3,560,448 | 22.09 | 0.1623 |
| DAGMM | 0.8723 | 120,648 | 121,679 | 2.77 | 0.0242 |
| RP | 0.5689 | 9,600 | 19,200 | 0.001 | 0.0043 |

**Table 5.18:** Cost-performance comparison on SMAP.

A notable observation is the strong performance of the simpler reconstruction-based approaches. On the SMD dataset, both PCA and Random Projections achieve performance comparable to several neural network-based methods when evaluating on window-level, despite their significantly lower computational complexity, results can be seen in Tables 5.1 and 5.2. Similarly, on SWaT, Random Projection outperform PCA and achieves performance levels that remain competitive with more sophisticated deep learning architectures (Tables 5.3 and 5.4). These results suggest that a considerable portion of anomaly structure in both datasets can be captured through a relatively simple low-dimensional representation.

The difference between PCA and Random Projection may be attributed to their respective objectives: PCA constructs a data-dependent linear projection that prioritises directions of maximum variance, whereas Random Projections preserve the overall geometric structure of the data without relying on variance estimates. In settings where anomalous behaviour is not aligned with dominant variance directions, this may lead PCA to discard information that remains preserved under Random Projections.

Among the neural network methods, variational and probabilistic approaches generally perform well across both datasets. The VAE achieves the highest performance in the SMD window-level setting, while OmniAnomaly achieves the highest overall performance on SWaT (Tables 5.1-5.4). This may indicate that explicitly modelling uncertainty in the latent space provides additional robustness when distinguishing between normal and anomalous behaviour. However, the performance gains relative to simple methods are modest, particularly compared to the additional computational cost for training.

KPCA is an interesting approach as it acts like a traditional method while capturing non-linear relationships. However, while KPCA performs competitively on SMD, its performance on SWaT is a lot worse than all other methods (Tables 5.1-5.4). As discussed previously, this is largely due to the aggressive subsampling required to make training computationally feasible. These results highlight one of the main

practical limitations of kernel-based approaches: although they are theoretically capable of modelling non-linear relationships, their computational complexity can become too expensive for large industrial datasets.

### 5.7.2 Stability

While most models achieved high clustering stability across datasets, high ARI does not necessarily imply meaningful cluster structure. A notable example is Omni-Anomaly on SMD, where perfect ARI is produced by DBSCAN assigning all points to a single cluster, a solution that is trivially stable but carries no important information. A different form of dissociation appears in TranAD on SMAP, which achieves the highest stability ARI (0.858) yet suffers one of the largest Silhouette score drops (0.256). This suggests that while the cluster assignment boundaries are robust to noise, the underlying cluster geometry is not. One possible explanation is that some of the models learn representations that are topologically consistent, preserving relative distances and cluster membership, without preserving the absolute geometry that determines cluster compactness. This motivates reporting both the ARI and Silhouette scores together, as either metric alone can give a misleading picture of representation robustness.

ARI and Silhouette score also disagree across several configurations. This disagreement reflects a fundamental difference between what the two metrics measure, which is interesting to note. A high Silhouette score with low ARI indicates that the model has organised the data into well-separated clusters, but according to some feature of the data other than the categories of interest, such as feature magnitude, periodicity or model-specific reconstruction geometry. This distinction matters for the intended use case. If the goal is to use clustering for anomaly characterisation or indication of root cause, ARI is the more relevant metric and Silhouette score alone is insufficient as a guide for selecting a model. In industrial settings where ground truth labels are rarely available, this becomes a practical constraint.

A recurring observation across datasets is that traditional models such as PCA and RP perform competitively with, and sometimes even outperform, more complex models. On SMD, RP matches VAE and AE with perfect stability ARI, and on the multi-machine SMD clustering task, PCA combined with UMAP approaches the best overall score (ARI 0.571 vs 0.614). This challenges the assumption that models trained with complex objectives, variational inference, adversarial training, and probabilistic latent variables necessarily produce richer or more structured representations. Anomaly clustering performance may depend more on how well a model preserves the global structure of the input than on its ability to model fine-grained temporal dynamics, and objectives that linear methods such as PCA are explicitly designed for. The gap may also reflect the fact that complex models are optimised for reconstruction or detection, objectives that do not directly reward cluster separability in latent space.

Interestingly, centroid-based methods (Fuzzy C-Means and K-Means) dominate as

the best performing clustering algorithm across models and datasets, with DBSCAN producing the best result in only isolated cases. Rather than reflecting a strict dominance of K-Means specifically, as mentioned in Section 6.3, this suggests that autoencoder representations tend to produce latent spaces with cluster geometry that centroid-based methods are well suited for. Reconstruction-based training implicitly discourages irregular or extreme encodings, producing a compact, bounded latent distribution. In the case of VAE, this regularisation is explicit, the KL divergence term directly penalises deviation from a unit Gaussian prior, creating a centroid-friendly latent geometry as an expected outcome. That FCM, as a soft generalisation of K-Means, performs comparably further supports this interpretation. The cases where DBSCAN outperforms centroid-based methods may correspond to the combination of model and dataset where latent geometry is denser or more irregularly shaped, but results indicate that these are exceptions. It is worth noting that fixing clustering hyperparameters across all models and noise levels may disadvantage DBSCAN disproportionately, since its performance is particularly sensitive to the choices of hyperparameters relative to the local density of a given latent space, as small mismatches can cause points to be classified as noise or cause distinct clusters to merge. The observed dominance of centroid-based methods should therefore be interpreted with this in mind.

### 5.7.3 Interpretability

This section discusses the interpretability results for the SMD dataset and the case study.

#### 5.7.3.1 SMD

The interpretability results on SMD demonstrate that the latent-space groupings produced by the LSTM, combined with UMAP and K-Means clustering, produce groupings that are largely explainable in terms of distinct feature-level patterns. Four of the six machines exhibit highly distinct profiles under both the cluster prototype $\bar{x}_A$ and the reconstruction-error profile $\bar{\mathbf{a}}_A$, with non-overlapping dominant features across clusters.

The mean pairwise cosine distance of 0.859 between cluster prototypes indicates that the clusters occupy largely orthogonal directions in feature space, meaning that each group is characterised by a different subset of features. This directly addresses RQ3: the latent-space groupings are not arbitrary but correspond to distinguishable behavioural patterns in the original input space. Machine-2-4, for example, is driven by feature F22 which appears in no other machine's top features, making its cluster immediately identifiable to an engineer inspecting the results.

The partial agreement between prototypes and error profiles (overlap ranging from 1/3 to 3/3 per machine) confirms that both measures are necessary for a complete assessment. Machine-2-2 shows perfect alignment with its anomalies being characterised by features F1–F3 in both the raw data and the model's reconstruction errors, indicating that these features are simultaneously abnormal in value *and* difficult for

the model to reconstruct.

The shared reconstruction failure on feature F9 between machines 1-7 and 3-6 (cosine distance 0.012) represents a limitation: while the latent space separates these machines geometrically (contributing to the overall ARI of 0.611), the model's *explanation* of why they are anomalous is indistinguishable. This suggests that F9 exhibits temporal dynamics that the LSTM consistently fails to capture, independent of machine identity. From a practical standpoint, an engineer inspecting an anomaly dominated by F9 would require additional context to determine its origin, while it can reduce the search space.

#### 5.7.3.2 Industrial Case Study

The results from the industrial case study reveal several findings regarding the applicability of learned representations to large-scale heterogeneous time-series data.

The discovered clusters do not align with the pseudo-labels derived from error-message similarity. The LSTM encoder groups runs by the similarity of their PM dynamics, which features are active, at what magnitudes, and with what temporal patterns, rather than by which error message was produced. Two runs that trigger the same textual error message may exhibit fundamentally different system behaviour, and conversely, runs from different pseudo-label clusters may share nearly identical PM dynamics. This suggests that error-message-based clustering captures *symptom* similarity, while PM-dynamics-based clustering captures *behavioural* similarity. For root-cause analysis, the latter may be more informative, as it reflects what the system was actually doing rather than what symptom it reported.

The feature categories that dominate the cluster prototypes $\bar{x}_A$ largely overlap with those dominating the reconstruction-error profiles $\bar{a}_A$ for the same cluster. This indicates consistency between the features that characterise a cluster's behaviour and the features the model finds difficult to reconstruct. In practical terms, the features that distinguish one failure mode from another are also the features that deviate most from the patterns learned during training, reinforcing their diagnostic relevance.

The LSTM architecture requires all input windows to have the same number of features, forcing the analysis to use only the common non-constant features across all runs ($\approx$ 2 800). Since individual test executions contain between 15 000 and $>$ 100 000 non-constant features, this restriction discards more than 97% of the available signal in the worst case. The information lost may include features that are uniquely active during specific failure modes and therefore highly discriminative. This represents a fundamental scalability limitation of cross-feature architectures when applied to heterogeneous industrial data, and motivates the exploration of architectures that can operate on variable-sized feature sets without requiring a shared feature vocabulary. In this context, models whose architecture does not depend on a fixed number of input features are of particular interest, as they can process each run's full feature set without requiring alignment to a shared feature

space.

### 5.7.4 Computational Trade-offs

The cost-performance results reveal a consistent pattern across all three datasets: the marginal accuracy gains in anomaly detection performance from complex architectures come at disproportionately higher computational cost.

Across the datasets, moving from simple linear methods such as PCA and RP to neural approaches provides performance improvements, indicating that nonlinear representations capture important structure that traditional methods miss. However, further increases in architectural complexity yield diminishing returns. Models with substantially larger parameter counts and computational requirements often produce only incremental improvements in detection performance, suggesting that complexity alone is not a reliable predictor of effectiveness.

The scaling behaviour across datasets further highlights these trade-offs. Traditional methods maintain negligible fitting costs regardless of dataset size, whereas neural approaches become increasingly expensive to train as sequence length and temporal complexity grow. This effect is particularly pronounced for Transformer-based architectures, whose self-attention mechanism scales poorly with longer sequences. As a result, methods that perform competitively on smaller datasets may become considerably less practical when deployed on larger industrial systems with respect to computational costs.

Inference costs are on the other hand less dramatic. All methods achieve submillisecond inference cost per sample, indicating that they would be suitable for deployment in real-time systems. However, differences that appear small in individual predictions can accumulate in high-throughput streaming system processing hundreds or thousands of observations each second.

These findings directly address RQ4: increased complexity does not uniformly justify its cost. The choice of method should be guided by operational constraints and performance requirements. When training resources are limited or frequent retraining is required, PCA or RP offer strong cost-effectiveness while retaining a large proportion of the achievable detection performance. When maximising detection accuracy is the primary objective and computational resources are available, sequence-based models such as LSTMs and OmniAnomaly provide the greatest benefit. Intermediate approaches such as AE, VAE, and USAD offer a practical compromise, achieving competitive performance without the computational burden of the most complex architectures.

The cost-performance relationship remains largely consistent when considering clustering rather than anomaly detection. On SMD, the LSTM encoder combined with UMAP achieves the highest clustering ARI (0.614), but PCA+UMAP follows closely at 0.572, a gap of 0.04 that comes at over 100× the training cost. On SMAP,

PCA+tSNE (0.397) trails the best model, OmniAnomaly (0.490), by a similar margin, and several complex architectures (VAE, AE, DAGMM) perform comparably to PCA. This reinforces the overall finding: increased model complexity provides modest gains across both detection and clustering tasks, and traditional methods remain competitive baselines relative to their computational cost. The exception is RP, which performs notably worse for clustering (ARI = 0.248 on SMD) despite strong detection and stability scores, suggesting that its geometry-preserving projection retains information sufficient for anomaly scoring but insufficient for separating distinct behavioural groups.

# 6

# Related Work

This chapter will describe related work in field and connect them to this project.

## 6.1 Reconstruction-based Anomaly Detection

Anomaly detection in multivariate time-series data is a well-studied problem within machine learning research. Reconstruction-based anomaly detection has emerged as a widely adopted approach, typically relying on models that learn representations of normal system behaviour and identify anomalies through reconstruction error when observed data deviate from learned patterns.

Several recent works have proposed specialised architectures for unsupervised anomaly detection in multivariate time-series settings. The following sections briefly describe a selection of influential approaches and discuss how their underlying ideas relate to the models and methodologies explored in this thesis.

### 6.1.1 DAGMM

DAGMM (Deep Autoencoding Gaussian Mixture Model) [61] is a joint optimization approach combining representation learning and density-based anomaly detection. It does so by learning a latent embedding of the input using an autoencoder while simultaneously fitting a Gaussian Mixture Model over the latent space to estimate the probability density of the observation.

The Gaussian Mixture Model plays a similar role to the latent-space regularisation in Variational Autoencoders, as described in Section 2.2.2. Both methods impose structure on the learned latent representation but while VAEs regularise the latent space towards a predefined prior distribution, DAGMM learns the latent density directly using the GMM.

Anomaly scores are then computed by combining the reconstruction error and the likelihood under the learned mixture model, i.e. how probable the latent representation is under the learned distribution of normal data. This allows the method to account for the structure of the latent space when predicting anomalies.

DAGMM was originally proposed as a general-purpose anomaly detection method and evaluated primarily on static tabular datasets. As a result, it does not explicitly model temporal dependencies, which limits its direct applicability to sequential data but makes it an important early baseline for combining reconstruction and latent-space density estimation approaches.

### 6.1.2 USAD

USAD (UnSupervised Anomaly Detection) [60] is a reconstruction-based learning approach that aims to improve the separation between normal and anomalous behaviour in the latent space by introducing an adversarial training architecture.

The model is a dual-autoencoder where one encoder feeds in to two separate decoders, $D_1$ and $D_2$, trained in an alternating fashion, $D_1$ is trained in the standard autoencoder fashion with an aim to minimise reconstruction error with respect to the input. $D_2$ is then trained using the original input combined with the reconstruction error of the first decoder, effectively learning to model the residual information not captured by $D_1$. This setup encourages the encoder to produce representations that are useful for reconstruction and sensitive to deviations from normal expected behaviour.

### 6.1.3 OmniAnomaly

OmniAnomaly [38] extends the reconstruction-based anomaly detection approach with recurrence, enabling the model to capture temporal dependencies using a stochastic recurrent variational autoencoder architecture.

The recurrence in the OmniAnomaly model adopts the Gated Recurrence Unit (GRU), which is a streamlined version of the LSTM states (described in Section 2.2.3.1) that merges the cell state and hidden state to a single vector using "reset" and "update" gates to decide what information is kept from past and passed along to future observations.

OmniAnomaly regularises the latent space produced by the encoder, producing a probability distribution for each observation rather than a deterministic embedding and encouraging the latent space towards a predefined prior distribution using KL-loss, as described in Section 2.2.2.

### 6.1.4 TranAD

TranAD [42] is a transformer-based architecture for multivariate time-series anomaly detection that leverages self-attention to capture long-range dependencies between datapoints. The attention mechanisms allows TranAD to model temporal relationships more efficiently as compared to recurrence based architectures like OmniAnomaly as it processes these simultaneously rather than sequentially, this attribute is further explained in Section 2.2.3.2.

Similar to USAD, TranAD introduces a multi-step reconstruction strategy, where an initial reconstruction is refined, but in this case using anomaly-aware attention mechanisms. This iterative approach improves the separation between normal and anomalous behaviour in the produced latent space by conditioning later reconstruction stages on information from earlier reconstruction errors. Unlike USAD which achieves this using a dual decoder approach, TranAD employs self-attention to dynamically attend to regions that are difficult to reconstruct, encouraging the model to emphasise anomalous temporal patterns.

## 6.2 Evaluations in Anomaly Detection

Selecting a suitable algorithm in time-series anomaly detection remains a challenge due to the diversity of anomaly types and data characteristics. Schmidl et al. [15] conducted a very thorough evaluation of 71 algorithms[1] across 976 datasets, concluding that no single algorithm consistently outperforms others across all scenarios. Their research highlights that while there is a trend toward complex deep learning architecture, these models are not yet competitive in terms of detection effectiveness despite their significantly higher computational cost.

Simpler models were instead found to often yield performance almost as good as the more sophisticated models. An example is the *DWT-MLEAD* algorithm, which was identified as having the best cost/benefit ratio, achieving an average AUC-ROC of 83% with an outstanding runtime of only 2.2ms per datapoint. This trend is also one we see in our evaluations, especially on the smaller datasets, where traditional linear projections like PCA and Random Projections achieved F1-scores very close to complex deep learning models like TransAE while requiring only a fraction of the training time. Our results on SMD further confirm this, showing that computationally heavy models, such as attention-based (TransAE) and LSTM-based architectures, sometimes performed worse in terms of $F1$ score and $MCC$ scores than the simpler baseline models. This supports the conclusion that increased model complexity does not necessarily translate into better detection accuracy, and fine-grained anomaly detection remains a significant challenge on smaller datasets, even for the more complex sequence models.

The study also identified that anomaly difficulty is heavily dictated by the type of anomaly. Obvious outliers, such as spikes in the data, are easy to detect, while anomalous trends pose a much bigger challenge for current algorithms. Additionally, anomalies in periodic sine waves are significantly easier to identify than those in chaotic structures, such as *Cylinder Bell Funnel* oscillations, which often lack enough structure for models to distinguish normal from abnormal behaviour. Finally, the study found that most algorithms are very sensitive to their parameter settings, requiring an average of seven specific settings to perform optimally, which underscores the importance of the stability metrics prioritised in this thesis.

[1]Note that all models included in our report, e.g. TranAD and USAD, are not included in this comprehensive evaluation.

## 6.3 The Role of Representation Learning

The evolution of time-series clustering has increasingly shifted toward representation learning-based methods to handle the complexities of high-dimensional data. According to a comprehensive survey by Paparrizos et al. [10], clustering techniques can be unified into a taxonomy consisting of four main groups: distance-based, distribution-based, subsequence-based, and representation learning-based methods. Traditional methods relying on Euclidean distance often suffer in the time-series domain because of their "lock-step" design, which requires a strict one-to-one mapping between data points in identical time indices. Because this design forces a point-by-point comparison, it cannot account for inherent distortions like scaling, shifting or noise. Representation learning addresses this by utilising a neural network to map raw input data into a lower-dimensional space. Specifically, generative-based approaches, such as the autoencoder architecture used in this research, learn robust representations by minimising reconstruction loss, which serves a dual purpose: Providing an anomaly score and enabling more effective downstream clustering.

A key objective in modern research is to discover latent spaces that are friendly to clustering. These spaces are referred to as "k-Means friendly" latent spaces and are discovered by simultaneously optimising autoencoder parameters and cluster centroids to ensure the latent structure is inherently suited for partitional grouping. Our findings contribute to this area by demonstrating that the latent space produced by an autoencoder becomes remarkably well-suited for k-means clustering, particularly when further refined using UMAP down to 2 or 3 dimensions. Notably, this suitability emerged as a natural property of the learned representation rather than through the explicit joint optimisation of centroids often required in existing literature. This approach also helps avoid the pitfalls of subsequence clustering, which prior studies warn can produce random or "meaningless" results unless very specific, difficult conditions are met. By learning a global representation before clustering, this thesis leverages these latent structures to identify meaningful, diagnostically useful patterns that are often masked by noise in the raw input data.

# 7

# Conclusion

In this thesis we have investigated whether learned representations can produce stable, interpretable, and diagnostically useful groupings of anomalous behaviour in high-dimensional multivariate time-series data. Eleven methods were evaluated spanning traditional linear projections (PCA, KPCA, RP), standard autoencoders (AE, VAE), temporal architectures (LSTM, TransAE), and specialised anomaly detection models (USAD, DAGMM, TranAD, OmniAnomaly), across three benchmark datasets and an industrial case study at Ericsson. We address each research question in turn and suggest directions for future work.

## 7.1 Summary of Findings

This section answers the research questions presented in Section 3.2, and concludes the most interesting findings.

### 7.1.1 RQ1: Reconstruction and Anomaly Detection

No single model consistently outperforms across datasets. OmniAnomaly and LSTM achieves the highest F1-scores across all datasets tested in this thesis. However, traditional methods remain competitive, achieving roughly 90% of the best F1-score across datasets at a fraction of the computational cost. This confirms recent findings that increased model complexity does not reliably translate into better detection [15].

### 7.1.2 RQ2: Stability

On well-structured datasets (SMD, SWaT), several methods achieve perfect or near-perfect clustering stability under perturbations. On SMAP, where the data exhibits less distinct cluster boundaries, no method achieves perfect stability. Simpler and more constrained models (RP, PCA, VAE) tend to produce the most stable representations, while more complex architectures (OmniAnomaly, DAGMM) are more sensitive to perturbations. The traditional methods PCA and RP achieves stability comparable to more complex models, indicating that preserving the geometric structure may be more important for clustering robustness than temporal modelling.

### 7.1.3 RQ3: Interpretability

The LSTM encoder combined with UMAP produces groupings that are largely interpretable: cluster prototypes $\bar{x}_A$ exhibit a mean pairwise cosine distance of 0.859, indicating that clusters are characterised by distinct feature subsets. In the industrial case study, the discovered clusters are dominated by distinct system components (coordination services, memory allocation, database metrics), demonstrating that the latent space captures structurally meaningful behavioural differences. As the method clusters the test executions based on PM dynamics rather than textual log pseudo-label, providing a complementary view to diagnose cause of failure. The consistency between cluster prototypes and reconstruction-error profiles reinforces the diagnostic relevance.

### 7.1.4 RQ4: Computational Trade-offs

While neural network architectures achieved the highest results across datasets, increased model complexity does not result in significant performance gain.

The relationship between model complexity and performance is non-linear. Initial gains from moving beyond linear projections are substantial, but further architectural complexity yields small returns for anomaly detection. For clustering, the additional cost of temporal models (LSTM) is better justified, as they produce latent spaces with meaningfully higher ARI. The choice of method should be guided by the primary downstream task: PCA or RP for cost-sensitive detection scenarios, and LSTM-based architectures when clustering quality is the primary concern.

### 7.1.5 Bridging Detection and Clustering

A key contribution of this work is demonstrating that reconstruction-based latent spaces are naturally suited for centroid-based clustering. This emerges as a by-product of the reconstruction objective, without requiring joint optimisation of cluster centroids. However, strong anomaly detection performance did not always lead to good clustering performance and thus neither task is a good predictor of a models performance on the other.

### 7.1.6 Industrial Applicability

The case study shows that the learned representations can capture meaningful behavioural patterns in heterogeneous, industrial high-dimensional time-series data. Although the discovered clusters do not align with the textual log-based pseudo-labels, they instead group test runs according to similarities in their performance metrics dynamics, suggesting value for behavioural analysis and supporting root-cause investigation. The study also reveals scalability limitations of proposed architectures. As the architectures require a shared input feature space across test runs, many available signals must be discarded when the feature space varies between executions. Traditional methods, that are agnostic to feature dimensionality, proved infeasible due to memory constraints. These finding highlight both the potential

of learned representations and the practical challenges that motivate the future research directions presented below.

## 7.2 Future Work

Given the conclusions we suggest two possible future directions of work:

- **Joint optimisation.** Exploring loss functions that simultaneously optimise for reconstruction quality and cluster separability in the latent space, potentially closing the gap between detection and clustering performance.

- **Variable-dimension architectures.** Developing or adapting models that handle variable feature sets without requiring alignment to a shared vocabulary, addressing the primary scalability limitation identified in the case study.

# Bibliography


[1] Umberto Michelucci. *Autoencoders*, pages 257–283. Apress, Berkeley, CA, 2022. ISBN 978-1-4842-8020-1. doi: 10.1007/978-1-4842-8020-1_9. URL https://doi.org/10.1007/978-1-4842-8020-1_9.

[2] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. URL https://D2L.ai.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[4] DBSCAN Explained: Unleashing the Power of Density-Based Clustering. Dbscan image. URL https://medium.com/@abhaysingh71711/dbscan-explained-unleashing-the-power-of-density-based-clustering-72a51ba40fdf. Medium article.

[5] Lai Leng Woo, Mark Zwolinski, and Basel Halak. Early detection of system-level anomalous behaviour using hardware performance counters. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 485–490, 2018. doi: 10.23919/DATE.2018.8342057.

[6] Joy Arulraj, Po-Chun Chang, Guoliang Jin, and Shan Lu. Production-run software failure diagnosis via hardware performance counters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, page 101112, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450318709. doi: 10.1145/2451116.2451128.

[7] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1285–1298. Association for Computing Machinery, 10 2017. ISBN

9781450349468. doi: 10.1145/3133956.3134015.

[8] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. Detecting large-scale system problems by mining console logs. In *SOSP'09 - Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles*, pages 117–131, 2009. ISBN 9781605587523. doi: 10.1145/1629575.1629587.

[9] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. Ts2vec: Towards universal representation of time series. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36: 8980–8987, 06 2022. doi: 10.1609/aaai.v36i8.20881.

[10] John Paparrizos, Fan Yang, and Haojun Li. Bridging the gap: A decade review of time-series clustering methods, 2024. URL `https://arxiv.org/abs/2412.20582`. arXiv preprint arXiv:2412.20582.

[11] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018*. The Internet Society, 2018. ISBN 1891562495. doi: 10.14722/ndss.2018.23204.

[12] Haixuan Guo, Shuhan Yuan, and Xintao Wu. LogBERT: Log Anomaly Detection via BERT. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2021-July. Institute of Electrical and Electronics Engineers Inc., 7 2021. ISBN 9780738133669. doi: 10.1109/IJCNN52387.2021.9534113.

[13] Linus Magnusson and Rasmus Thorsson. Clue clustering-based load understanding and exploration. Master's thesis, Göteborgs Universitet, 2025. URL `https://hdl.handle.net/2077/89783`.

[14] Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou, and Philippas Tsigas. Ip.lsh.dbscan: Integrated parallel density-based clustering by locality-sensitive hashing. *Discrete Applied Mathematics*, 382:183–196, 2026. ISSN 0166-218X. doi: https://doi.org/10.1016/j.dam.2025.11.047.

[15] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: a comprehensive evaluation. *Proc. VLDB Endow.*, 15(9): 17791797, May 2022. ISSN 2150-8097. doi: 10.14778/3538598.3538602. URL `https://doi.org/10.14778/3538598.3538602`.

[16] Jens E. d'Hondt, Haojun Li, Fan Yang, Odysseas Papapetrou, and John Paparrizos. A structured study of multivariate time-series distance measures. *Proceedings of the ACM on Management of Data*, 3(3):1–29, 6 2025. doi: 10.1145/3725258.

[17] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*,

IMC '03, page 234247, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137737. doi: 10.1145/948205.948236. URL https://doi.org/10.1145/948205.948236.

[18] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *ACM International Conference Proceeding Series*, volume 382, 2009. ISBN 9781605585161. doi: 10.1145/1553374.1553516.

[19] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 2006, pages 287–296. Association for Computing Machinery (ACM), 2006. ISBN 1595933395. doi: 10.1145/1150402.1150436.

[20] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:498–520, 1933. doi: https://doi.org/10.1037/h0071325.

[21] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 1*, 2:559–572, 1901. doi: https://doi.org/10.1080/14786440109462720.

[22] William B. Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into hilbert space. *Contemporary mathematics*, 26:189–206, 1984. URL https://api.semanticscholar.org/CorpusID:117819162.

[23] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, page 245250, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 158113391X. doi: 10.1145/502512.502546. URL https://doi.org/10.1145/502512.502546.

[24] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991. doi: https://doi.org/10.1002/aic.690370209.

[25] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10 (5):1299–1319, 1998. doi: 10.1162/089976698300017467.

[26] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

[27] Leland McInnes, John Healy, Nathaniel Saul, and Lukas GroSSberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018. doi: 10.21105/joss.00861. URL https://doi.org/10.2

1105/joss.00861.

[28] Muhammad Ali, Mark W. Jones, Xianghua Xie, and Eleni Vasilaki. Timecluster: dimension reduction applied to temporal data for visual analytics. *The Visual Computer*, 35(6):1013–1026, 2019. doi: 10.1007/s00371-019-01673-y. URL https://doi.org/10.1007/s00371-019-01673-y.

[29] Etienne Becht, Leland McInnes, John Healy, Charles-Antoine Dutertre, Iain W. H. Kwok, Lai Guan Ng, Florent Ginhoux, and Evan W. Newell. Dimensionality reduction for visualizing single-cell data using umap. *Nature Biotechnology*, 37(1):38–44, 2019. doi: 10.1038/nbt.4314. URL https://doi.org/10.1038/nbt.4314.

[30] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[31] Laurens van der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of Machine Learning Research*, 15(93):3221–3245, 2014. URL http://jmlr.org/papers/v15/vandermaaten14a.html.

[32] George C. Linderman, Manas Rachh, Jared G. Hoskins, Stefan Steinerberger, and Yuval Kluger. Fast interpolation-based t-sne for improved visualization of single-cell rna-seq data. *Nature Methods*, 16:243–245, February 2019. doi: 10.1038/s41592-018-0308-4. URL https://doi.org/10.1038/s41592-018-0308-4.

[33] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35 (8):17981828, August 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2013.50. URL https://doi.org/10.1109/TPAMI.2013.50.

[34] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pages 318–362. MIT Press, 1988.

[35] Benyamin Ghojogh, Mark Crowley, Fakhri Karray, and Ali Ghodsi. *Elements of Dimensionality Reduction and Manifold Learning*. Springer Cham, 2023. ISBN 978-3-031-10602-6. doi: 10.1007/978-3-031-10602-6.

[36] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017. URL https://arxiv.org/abs/1710.05941.

[37] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trendső in Machine Learning*, 12(4):307392, November 2019. ISSN 1935-8245. doi: 10.1561/2200000056. URL http://dx.doi.org/10.1561/2200000056.

[38] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan 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*, KDD '19, page 28282837, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330672.

[39] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection, 2016. URL https://arxiv.org/abs/1607.00148.

[40] Tianming Xie, Qifa Xu, and Cuixia Jiang. Anomaly detection for multivariate times series through the multi-scale convolutional recurrent variational autoencoder. *Expert Systems with Applications*, 231:120725, 2023. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2023.120725.

[41] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using lstms and non-parametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 387395, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219845.

[42] Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. Tranad: deep transformer networks for anomaly detection in multivariate time series data. *Proc. VLDB Endow.*, 15(6):12011214, February 2022. ISSN 2150-8097. doi: 10.14778/3514061.3514067. URL https://doi.org/10.14778/3514061.3514067.

[43] Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Anomaly transformer: Time series anomaly detection with association discrepancy. In *International Conference on Learning Representations*, 2022. doi: https://doi.org/10.48550/arXiv.2110.02642. Workshop presentation: https://iclr.cc/virtual/2022/spotlight/7024.

[44] Chong Zhou and Randy C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 665674, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3098052. URL https://doi.org/10.1145/3097983.3098052.

[45] Jung Kyu Seo, JuHyeon Lee, Buyoung Kim, Wooseong Shim, and Jung Taek Seo. Ai-based anomaly detection in industrial control and cyberphysical systems: A data-type-oriented systematic review. *Electronics*, 15(1), 2026. ISSN 2079-9292. doi: 10.3390/electronics15010020. URL https://www.mdpi.com/2079-9292/15/1/20.

[46] Z. Li, Y. Yan, X. Wang, et al. A survey of deep learning for industrial visual anomaly detection. *Artificial Intelligence Review*, 58:279, 2025. doi: 10.1007/s10462-025-11287-7. URL https://doi.org/10.1007/s10462-025-11287-7.

Accepted: 30 May 2025; Published: 14 June 2025.

[47] Stuart Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer Series in Statistics. Springer London, 1 edition, 2001. ISBN 978-1-85233-459-8. doi: 10.1007/978-1-4471-3675-0. URL https://doi.org/10.1007/978-1-4471-3675-0.

[48] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 10671075, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3098144. URL https://doi.org/10.1145/3097983.3098144.

[49] Arvid Naess. *The Peaks-Over-Threshold Method*, pages 19–28. Springer Nature Switzerland, Cham, 2024. ISBN 978-3-031-60769-1. doi: 10.1007/978-3-031-60769-1_3. URL https://doi.org/10.1007/978-3-031-60769-1_3.

[50] David MW Powers. Evaluation: From precision, recall and f-measure to roc, informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[51] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264323, September 1999. ISSN 0360-0300. doi: 10.1145/331499.331504. URL https://doi.org/10.1145/331499.331504.

[52] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei 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*, KDD'96, page 226231. AAAI Press, 1996. URL https://dl.acm.org/doi/10.5555/3001460.3001507.

[53] J A Hartigan and M A Wong. Algorithm AS 136: A K-Means Clustering Algorithm. *Source: Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979. doi: https://doi.org/10.2307/2346830.

[54] James C. Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers Geosciences*, 10(2):191–203, 1984. ISSN 0098-3004. doi: https://doi.org/10.1016/0098-3004(84)90020-7. URL https://www.sciencedirect.com/science/article/pii/0098300484900207.

[55] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, page 518529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1558606157. URL https://www.vldb.org/conf/1999/P49.pdf.

[56] Wenhao Wu, Weiwei Wang, Xixi Jia, and Xiangchu Feng. Transformer autoencoder for k-means efficient clustering. *Engineering Applications of Artificial*

*Intelligence*, 133:108612, 2024. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2024.108612.

[57] WASP. WARA Ops. https://wasp-sweden.org/research/research-arenas/wara-operational-data/, 2026. Accessed: 2026-07-03.

[58] Jens E. dHondt, Koen Minartz, and Odysseas Papapetrou. Efficient detection of multivariate correlations with different correlation measures. *VLDB Journal*, 33(2):481–505, 3 2024. ISSN 0949877X. doi: 10.1007/s00778-023-00815-y.

[59] Dino Ienco and Roberto Interdonato. Deep Multivariate Time Series Embedding Clustering via Attentive-Gated Autoencoder. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12084 LNAI, pages 318–329. Springer, 2020. ISBN 9783030474256. doi: https://doi.org/10.1007/978-3-030-47426-3_25.

[60] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 33953404, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403392. URL https://doi.org/10.1145/3394486.3403392.

[61] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Dae ki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018. URL https://api.semanticscholar.org/CorpusID:51805340.

[62] https://www.kaggle.com/datasets/vishala28/swat-dataset-secure-water-treatment-system?select=attack.csv. URL https://itrust.sutd.edu.sg/itrust-labs_datasets/. SWaT - Secure Water Treatment, iTrust, Centre for Research in Cyber Security, Singapore University of Technology and Design.

[63] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan 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*, KDD '19, page 28282837, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330672. URL https://doi.org/10.1145/3292500.3330672.

[64] Kim Siwon, Kukjin Choi, Hyun-Soo Choi, Byunghan Lee, and Sungroh Yoon. Towards a rigorous evaluation of time-series anomaly detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36:7194–7201, 06 2022. doi: 10.1609/aaai.v36i7.20680.

[65] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985. doi: 10.1007/BF01908075.

[66] Pablo A. Estévez, Michel Tesmer, Claudio A. Perez, and Jacek M. Zurada. Normalized mutual information feature selection. *Trans. Neur. Netw.*, 20(2): 189201, February 2009. ISSN 1045-9227. doi: 10.1109/TNN.2008.2005601. URL `https://doi.org/10.1109/TNN.2008.2005601`.

[67] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7.

[68] Selecting the number of clusters with silhouette analysis on KMeans clustering. URL `https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html`. Scikit-learn documentation.

[69] Ketan Rajshekhar Shahapure and Charles Nicholas. Cluster quality analysis using silhouette score. In *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 747–748, 2020. doi: 10.1109/DSAA49011.2020.00096.

[70] Ulrike von Luxburg. Clustering stability: An overview. *Foundations and Trendső in Machine Learning*, 2(3):235274, March 2010. ISSN 1935-8237. doi: 10.1561/2200000008.

[71] Pranjal Awasthi, Avrim Blum, and Or Sheffet. Center-based clustering under perturbation stability. *Information Processing Letters*, 112(1):49–54, 2012. ISSN 0020-0190. doi: https://doi.org/10.1016/j.ipl.2011.10.006.

[72] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. Deep anomaly detection on attributed networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 594–602, 05 2019. ISBN 978-1-61197-567-3. doi: 10.1137/1.9781611975673.67.

[73] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 47684777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964. doi: 10.5555/3295222.3295230.

[74] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 387395, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219845.

[75] NASA Jet Propulsion Laboratory. Soil Moisture Active Passive (SMAP) Mis-

sion. https://smap.jpl.nasa.gov/, 2026. Accessed: 2026-03-25.

[76] Lukas Biewald. Experiment tracking with weights and biases, 2026. URL https://www.wandb.com/. Software available from wandb.com.

[77] Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. doi: 10.5555/2999325.2999464. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf.

# 8

# Appendix 1

The exhaustive tables below shows the results of all combinations of clustering methods and autoencoder architectures. 4 different noise-levels were tried with $\sigma \in \{0.1, 0.5, 1.0, 2.0\}$ to see how robust the different methods were under Gaussian noise.

**Table 8.1:** Full evaluation of clustering stability across models at different noise levels on SWaT dataset

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| PCA | KMeans | 0.10 | 0.996 | 0.006 | 0.949 | 0.949 |
| PCA | KMeans | 0.50 | 0.993 | 0.007 | 0.949 | 0.949 |
| PCA | KMeans | 1.00 | 0.992 | 0.006 | 0.949 | 0.949 |
| PCA | KMeans | 2.00 | 0.992 | 0.006 | 0.949 | 0.949 |
| PCA | FCM | 0.10 | 1.000 | 0.000 | 0.580 | 0.580 |
| PCA | FCM | 0.50 | 1.000 | 0.000 | 0.580 | 0.579 |
| PCA | FCM | 1.00 | 1.000 | 0.000 | 0.580 | 0.577 |
| PCA | FCM | 2.00 | 1.000 | 0.000 | 0.580 | 0.570 |
| PCA | DBSCAN | 0.10 | 0.999 | 0.004 | 0.953 | 0.953 |
| PCA | DBSCAN | 0.50 | 0.994 | 0.006 | 0.953 | 0.951 |
| PCA | DBSCAN | 1.00 | 0.994 | 0.006 | 0.953 | 0.951 |
| PCA | DBSCAN | 2.00 | 0.994 | 0.006 | 0.953 | 0.951 |
| KPCA | KMeans | 0.10 | 1.000 | 0.000 | 0.904 | 0.903 |
| KPCA | KMeans | 0.50 | 1.000 | 0.001 | 0.904 | 0.899 |
| KPCA | KMeans | 1.00 | 0.999 | 0.001 | 0.904 | 0.889 |
| KPCA | KMeans | 2.00 | 0.996 | 0.002 | 0.904 | 0.850 |
| KPCA | FCM | 0.10 | 1.000 | 0.000 | 0.903 | 0.903 |
| KPCA | FCM | 0.50 | 0.999 | 0.001 | 0.903 | 0.899 |
| KPCA | FCM | 1.00 | 0.998 | 0.002 | 0.903 | 0.888 |
| KPCA | FCM | 2.00 | 0.990 | 0.005 | 0.903 | 0.832 |
| KPCA | DBSCAN | 0.10 | 1.000 | 0.000 | 0.878 | 0.879 |
| KPCA | DBSCAN | 0.50 | 0.989 | 0.001 | 0.878 | 0.874 |
| KPCA | DBSCAN | 1.00 | 0.505 | 0.000 | 0.878 | 0.622 |
| KPCA | DBSCAN | 2.00 | 0.000 | 0.000 | 0.878 | nan |

Continued on next page

Table 8.1 – continued from previous page

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| RP | KMeans | 0.10 | 1.000 | 0.000 | 0.972 | 0.972 |
| RP | KMeans | 0.50 | 1.000 | 0.000 | 0.972 | 0.971 |
| RP | KMeans | 1.00 | 1.000 | 0.000 | 0.972 | 0.970 |
| RP | KMeans | 2.00 | 0.999 | 0.004 | 0.972 | 0.965 |
| RP | FCM | 0.10 | 0.979 | 0.008 | 0.349 | 0.348 |
| RP | FCM | 0.50 | 0.958 | 0.008 | 0.349 | 0.317 |
| RP | FCM | 1.00 | 0.878 | 0.023 | 0.349 | 0.266 |
| RP | FCM | 2.00 | 0.490 | 0.038 | 0.349 | 0.088 |
| RP | DBSCAN | 0.10 | 1.000 | 0.000 | 0.955 | 0.955 |
| RP | DBSCAN | 0.50 | 1.000 | 0.000 | 0.955 | 0.954 |
| RP | DBSCAN | 1.00 | 1.000 | 0.000 | 0.955 | 0.952 |
| RP | DBSCAN | 2.00 | 1.000 | 0.000 | 0.955 | 0.946 |
| AE | KMeans | 0.10 | 1.000 | 0.000 | 0.943 | 0.943 |
| AE | KMeans | 0.50 | 0.995 | 0.010 | 0.943 | 0.945 |
| AE | KMeans | 1.00 | 0.992 | 0.012 | 0.943 | 0.946 |
| AE | KMeans | 2.00 | 0.980 | 0.011 | 0.943 | 0.949 |
| AE | FCM | 0.10 | 0.998 | 0.001 | 0.398 | 0.398 |
| AE | FCM | 0.50 | 0.977 | 0.006 | 0.398 | 0.396 |
| AE | FCM | 1.00 | 0.956 | 0.011 | 0.398 | 0.392 |
| AE | FCM | 2.00 | 0.904 | 0.014 | 0.398 | 0.380 |
| AE | DBSCAN | 0.10 | 0.995 | 0.006 | 0.945 | 0.944 |
| AE | DBSCAN | 0.50 | 0.995 | 0.006 | 0.945 | 0.944 |
| AE | DBSCAN | 1.00 | 0.995 | 0.006 | 0.945 | 0.943 |
| AE | DBSCAN | 2.00 | 0.995 | 0.006 | 0.945 | 0.943 |
| LSTM | KMeans | 0.50 | 0.998 | 0.002 | 0.549 | 0.544 |
| LSTM | KMeans | 1.00 | 0.984 | 0.004 | 0.549 | 0.534 |
| LSTM | KMeans | 2.00 | 0.973 | 0.019 | 0.549 | 0.498 |
| LSTM | KMeans | 0.10 | 0.995 | 0.014 | 0.549 | 0.549 |
| LSTM | FCM | 0.10 | 1.000 | 0.000 | 0.515 | 0.515 |
| LSTM | FCM | 0.50 | 0.997 | 0.002 | 0.515 | 0.510 |
| LSTM | FCM | 1.00 | 0.991 | 0.002 | 0.515 | 0.497 |
| LSTM | FCM | 2.00 | 0.965 | 0.042 | 0.515 | 0.443 |
| LSTM | DBSCAN | 0.10 | 0.999 | 0.001 | 0.594 | 0.593 |
| LSTM | DBSCAN | 0.50 | 0.997 | 0.001 | 0.594 | 0.580 |
| LSTM | DBSCAN | 1.00 | 0.985 | 0.004 | 0.594 | 0.535 |
| LSTM | DBSCAN | 2.00 | 0.755 | 0.103 | 0.594 | 0.363 |
| TransAE | KMeans | 0.10 | 0.998 | 0.001 | 0.605 | 0.605 |
| TransAE | KMeans | 0.50 | 0.997 | 0.001 | 0.605 | 0.598 |
| TransAE | KMeans | 1.00 | 0.993 | 0.006 | 0.605 | 0.582 |
| TransAE | KMeans | 2.00 | 0.930 | 0.023 | 0.605 | 0.545 |
| TransAE | FCM | 0.10 | 1.000 | 0.000 | 0.601 | 0.600 |
| TransAE | FCM | 0.50 | 0.930 | 0.103 | 0.601 | 0.552 |

Continued on next page

Table 8.1 – continued from previous page

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| TransAE | FCM | 1.00 | 0.948 | 0.092 | 0.601 | 0.550 |
| TransAE | FCM | 2.00 | 0.733 | 0.007 | 0.601 | 0.384 |
| TransAE | DBSCAN | 0.10 | 1.000 | 0.000 | 0.661 | 0.661 |
| TransAE | DBSCAN | 0.50 | 0.953 | 0.124 | 0.661 | 0.634 |
| TransAE | DBSCAN | 1.00 | 0.779 | 0.208 | 0.661 | 0.560 |
| TransAE | DBSCAN | 2.00 | 0.546 | 0.004 | 0.661 | 0.435 |
| VAE | KMeans | 0.10 | 1.000 | 0.000 | 0.962 | 0.962 |
| VAE | KMeans | 0.50 | 1.000 | 0.000 | 0.962 | 0.962 |
| VAE | KMeans | 1.00 | 1.000 | 0.000 | 0.962 | 0.962 |
| VAE | KMeans | 2.00 | 1.000 | 0.000 | 0.962 | 0.962 |
| VAE | FCM | 0.10 | 1.000 | 0.000 | 0.411 | 0.410 |
| VAE | FCM | 0.50 | 0.998 | 0.002 | 0.411 | 0.409 |
| VAE | FCM | 1.00 | 0.994 | 0.002 | 0.411 | 0.404 |
| VAE | FCM | 2.00 | 0.988 | 0.002 | 0.411 | 0.390 |
| VAE | DBSCAN | 0.10 | 1.000 | 0.000 | 0.958 | 0.958 |
| VAE | DBSCAN | 0.50 | 1.000 | 0.000 | 0.958 | 0.958 |
| VAE | DBSCAN | 1.00 | 1.000 | 0.000 | 0.958 | 0.958 |
| VAE | DBSCAN | 2.00 | 1.000 | 0.000 | 0.958 | 0.957 |
| DAGMM | KMeans | 0.10 | 0.995 | 0.004 | 0.639 | 0.638 |
| DAGMM | KMeans | 0.50 | 0.983 | 0.006 | 0.639 | 0.620 |
| DAGMM | KMeans | 1.00 | 0.974 | 0.008 | 0.639 | 0.580 |
| DAGMM | KMeans | 2.00 | 0.930 | 0.010 | 0.639 | 0.486 |
| DAGMM | FCM | 0.10 | 0.933 | 0.095 | 0.582 | 0.564 |
| DAGMM | FCM | 0.50 | 0.889 | 0.060 | 0.582 | 0.528 |
| DAGMM | FCM | 1.00 | 0.854 | 0.020 | 0.582 | 0.501 |
| DAGMM | FCM | 2.00 | 0.711 | 0.106 | 0.582 | 0.419 |
| DAGMM | DBSCAN | 0.10 | 0.998 | 0.005 | 0.467 | 0.464 |
| DAGMM | DBSCAN | 0.50 | 0.971 | 0.070 | 0.467 | 0.440 |
| DAGMM | DBSCAN | 1.00 | 0.877 | 0.122 | 0.467 | 0.380 |
| DAGMM | DBSCAN | 2.00 | 0.731 | 0.067 | 0.467 | 0.266 |
| OmniAnomaly | KMeans | 0.10 | 0.995 | 0.003 | 0.463 | 0.458 |
| OmniAnomaly | KMeans | 0.50 | 0.984 | 0.006 | 0.463 | 0.389 |
| OmniAnomaly | KMeans | 1.00 | 0.493 | 0.063 | 0.463 | 0.281 |
| OmniAnomaly | KMeans | 2.00 | 0.470 | 0.011 | 0.463 | 0.202 |
| OmniAnomaly | FCM | 0.10 | 0.991 | 0.002 | 0.440 | 0.436 |
| OmniAnomaly | FCM | 0.50 | 0.865 | 0.121 | 0.440 | 0.373 |
| OmniAnomaly | FCM | 1.00 | 0.646 | 0.013 | 0.440 | 0.249 |
| OmniAnomaly | FCM | 2.00 | 0.521 | 0.012 | 0.440 | 0.185 |
| OmniAnomaly | DBSCAN | 0.10 | 0.606 | 0.017 | −0.289 | −0.295 |
| OmniAnomaly | DBSCAN | 0.50 | 0.096 | 0.000 | −0.289 | nan |
| OmniAnomaly | DBSCAN | 1.00 | 0.096 | 0.000 | −0.289 | nan |
| OmniAnomaly | DBSCAN | 2.00 | 0.096 | 0.000 | −0.289 | nan |

Continued on next page

Table 8.1 – continued from previous page

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| USAD | KMeans | 0.10 | 0.769 | 0.354 | 0.968 | 0.899 |
| USAD | KMeans | 0.50 | 0.692 | 0.378 | 0.968 | 0.875 |
| USAD | KMeans | 1.00 | 0.846 | 0.308 | 0.968 | 0.920 |
| USAD | KMeans | 2.00 | 0.923 | 0.230 | 0.968 | 0.943 |
| USAD | FCM | 0.10 | 0.998 | 0.002 | 0.736 | 0.736 |
| USAD | FCM | 0.50 | 0.992 | 0.004 | 0.736 | 0.734 |
| USAD | FCM | 1.00 | 0.988 | 0.005 | 0.736 | 0.729 |
| USAD | FCM | 2.00 | 0.978 | 0.005 | 0.736 | 0.714 |
| USAD | DBSCAN | 0.10 | 0.976 | 0.009 | 0.798 | 0.825 |
| USAD | DBSCAN | 0.50 | 0.871 | 0.033 | 0.798 | 0.847 |
| USAD | DBSCAN | 1.00 | 0.762 | 0.036 | 0.798 | 0.839 |
| USAD | DBSCAN | 2.00 | 0.652 | 0.030 | 0.798 | 0.829 |
| TranAD | KMeans | 0.10 | 1.000 | 0.000 | 0.951 | 0.951 |
| TranAD | KMeans | 0.50 | 1.000 | 0.000 | 0.951 | 0.949 |
| TranAD | KMeans | 1.00 | 0.982 | 0.020 | 0.951 | 0.950 |
| TranAD | KMeans | 2.00 | 0.976 | 0.008 | 0.951 | 0.957 |
| TranAD | FCM | 0.10 | 1.000 | 0.000 | 0.613 | 0.612 |
| TranAD | FCM | 0.50 | 0.997 | 0.001 | 0.613 | 0.606 |
| TranAD | FCM | 1.00 | 0.997 | 0.000 | 0.613 | 0.595 |
| TranAD | FCM | 2.00 | 0.995 | 0.002 | 0.613 | 0.566 |
| TranAD | DBSCAN | 0.10 | 1.000 | 0.000 | 0.740 | 0.740 |
| TranAD | DBSCAN | 0.50 | 0.999 | 0.000 | 0.740 | 0.736 |
| TranAD | DBSCAN | 1.00 | 0.997 | 0.007 | 0.740 | 0.728 |
| TranAD | DBSCAN | 2.00 | 0.988 | 0.012 | 0.740 | 0.714 |

**Table 8.2:** Full evaluation of clustering stability across all models at different noise levels on SMD (machine-1-1)

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| PCA | KMeans | 0.10 | 0.992 | 0.002 | 0.631 | 0.630 |
| PCA | KMeans | 0.50 | 0.958 | 0.008 | 0.631 | 0.601 |
| PCA | KMeans | 1.00 | 0.929 | 0.005 | 0.631 | 0.527 |
| PCA | KMeans | 2.00 | 0.853 | 0.010 | 0.631 | 0.371 |
| PCA | FCM | 0.10 | 0.981 | 0.003 | 0.346 | 0.342 |
| PCA | FCM | 0.50 | 0.842 | 0.007 | 0.346 | 0.250 |
| PCA | FCM | 1.00 | 0.296 | 0.014 | 0.346 | 0.051 |
| PCA | FCM | 2.00 | 0.197 | 0.003 | 0.346 | 0.200 |
| PCA | DBSCAN | 0.10 | 0.739 | 0.024 | 0.380 | 0.369 |
| PCA | DBSCAN | 0.50 | 0.485 | 0.035 | 0.380 | nan |
| PCA | DBSCAN | 1.00 | −0.052 | 0.002 | 0.380 | nan |
| PCA | DBSCAN | 2.00 | 0.000 | 0.000 | 0.380 | nan |
| KPCA | KMeans | 0.10 | 0.984 | 0.002 | 0.378 | 0.373 |
| KPCA | KMeans | 0.50 | 0.916 | 0.010 | 0.378 | 0.302 |

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| KPCA | KMeans | 1.00 | 0.760 | 0.039 | 0.378 | 0.191 |
| KPCA | KMeans | 2.00 | 0.390 | 0.021 | 0.378 | 0.090 |
| KPCA | FCM | 0.10 | 0.971 | 0.003 | 0.325 | 0.321 |
| KPCA | FCM | 0.50 | 0.814 | 0.009 | 0.325 | 0.246 |
| KPCA | FCM | 1.00 | 0.437 | 0.008 | 0.325 | 0.113 |
| KPCA | FCM | 2.00 | 0.225 | 0.006 | 0.325 | 0.147 |
| KPCA | DBSCAN | 0.10 | 0.839 | 0.014 | 0.058 | −0.021 |
| KPCA | DBSCAN | 0.50 | 0.023 | 0.000 | 0.058 | nan |
| KPCA | DBSCAN | 1.00 | 0.024 | 0.001 | 0.058 | nan |
| KPCA | DBSCAN | 2.00 | 0.000 | 0.000 | 0.058 | nan |
| RP | KMeans | 0.10 | 1.000 | 0.000 | 0.998 | 0.998 |
| RP | KMeans | 0.50 | 1.000 | 0.000 | 0.998 | 0.997 |
| RP | KMeans | 1.00 | 1.000 | 0.000 | 0.998 | 0.996 |
| RP | KMeans | 2.00 | 1.000 | 0.000 | 0.998 | 0.995 |
| RP | FCM | 0.10 | 1.000 | 0.000 | 0.997 | 0.997 |
| RP | FCM | 0.50 | 1.000 | 0.000 | 0.997 | 0.996 |
| RP | FCM | 1.00 | 0.041 | 0.021 | 0.997 | −0.268 |
| RP | FCM | 2.00 | 0.010 | 0.000 | 0.997 | 0.063 |
| RP | DBSCAN | 0.10 | 1.000 | 0.000 | nan | nan |
| RP | DBSCAN | 0.50 | 1.000 | 0.000 | nan | nan |
| RP | DBSCAN | 1.00 | 1.000 | 0.000 | nan | nan |
| RP | DBSCAN | 2.00 | 1.000 | 0.000 | nan | nan |
| AE | KMeans | 0.10 | 1.000 | 0.000 | 0.995 | 0.995 |
| AE | KMeans | 0.50 | 1.000 | 0.000 | 0.995 | 0.995 |
| AE | KMeans | 1.00 | 1.000 | 0.000 | 0.995 | 0.995 |
| AE | KMeans | 2.00 | 1.000 | 0.000 | 0.995 | 0.994 |
| AE | FCM | 0.10 | 0.993 | 0.002 | 0.772 | 0.772 |
| AE | FCM | 0.50 | 0.968 | 0.006 | 0.772 | 0.770 |
| AE | FCM | 1.00 | 0.935 | 0.006 | 0.772 | 0.762 |
| AE | FCM | 2.00 | 0.845 | 0.010 | 0.772 | 0.729 |
| AE | DBSCAN | 0.10 | 1.000 | 0.000 | nan | nan |
| AE | DBSCAN | 0.50 | 1.000 | 0.000 | nan | nan |
| AE | DBSCAN | 1.00 | 1.000 | 0.000 | nan | nan |
| AE | DBSCAN | 2.00 | 1.000 | 0.000 | nan | nan |
| LSTM | KMeans | 0.10 | 0.985 | 0.002 | 0.459 | 0.455 |
| LSTM | KMeans | 0.50 | 0.922 | 0.004 | 0.459 | 0.394 |
| LSTM | KMeans | 1.00 | 0.557 | 0.153 | 0.459 | 0.252 |
| LSTM | KMeans | 2.00 | 0.322 | 0.010 | 0.459 | 0.192 |
| LSTM | FCM | 0.10 | 0.927 | 0.004 | 0.316 | 0.313 |
| LSTM | FCM | 0.50 | 0.703 | 0.006 | 0.316 | 0.270 |
| LSTM | FCM | 1.00 | 0.515 | 0.005 | 0.316 | 0.219 |
| LSTM | FCM | 2.00 | 0.269 | 0.005 | 0.316 | 0.123 |
| LSTM | DBSCAN | 0.10 | 0.893 | 0.004 | 0.150 | 0.215 |
| LSTM | DBSCAN | 0.50 | 0.084 | 0.008 | 0.150 | −0.143 |
| LSTM | DBSCAN | 1.00 | −0.073 | 0.005 | 0.150 | −0.327 |

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| LSTM | DBSCAN | 2.00 | 0.000 | 0.000 | 0.150 | nan |
| TransAE | KMeans | 0.10 | 0.974 | 0.002 | 0.489 | 0.487 |
| TransAE | KMeans | 0.50 | 0.827 | 0.008 | 0.489 | 0.390 |
| TransAE | KMeans | 1.00 | 0.366 | 0.022 | 0.489 | 0.235 |
| TransAE | KMeans | 2.00 | 0.165 | 0.008 | 0.489 | 0.168 |
| TransAE | FCM | 0.10 | 0.910 | 0.005 | 0.319 | 0.311 |
| TransAE | FCM | 0.50 | 0.514 | 0.005 | 0.319 | 0.251 |
| TransAE | FCM | 1.00 | 0.300 | 0.008 | 0.319 | 0.178 |
| TransAE | FCM | 2.00 | 0.111 | 0.004 | 0.319 | 0.061 |
| TransAE | DBSCAN | 0.10 | 0.820 | 0.170 | −0.113 | 0.041 |
| TransAE | DBSCAN | 0.50 | 0.230 | 0.014 | −0.113 | 0.316 |
| TransAE | DBSCAN | 1.00 | 0.148 | 0.006 | −0.113 | 0.462 |
| TransAE | DBSCAN | 2.00 | 0.134 | 0.004 | −0.113 | 0.500 |
| VAE | KMeans | 0.10 | 1.000 | 0.000 | 0.998 | 0.998 |
| VAE | KMeans | 0.50 | 1.000 | 0.000 | 0.998 | 0.997 |
| VAE | KMeans | 1.00 | 1.000 | 0.000 | 0.998 | 0.996 |
| VAE | KMeans | 2.00 | 1.000 | 0.000 | 0.998 | 0.995 |
| VAE | FCM | 0.10 | 1.000 | 0.000 | 0.998 | 0.998 |
| VAE | FCM | 0.50 | 1.000 | 0.000 | 0.998 | 0.997 |
| VAE | FCM | 1.00 | 0.006 | 0.000 | 0.998 | 0.219 |
| VAE | FCM | 2.00 | 0.007 | 0.000 | 0.998 | 0.376 |
| VAE | DBSCAN | 0.10 | 1.000 | 0.000 | nan | nan |
| VAE | DBSCAN | 0.50 | 1.000 | 0.000 | nan | nan |
| VAE | DBSCAN | 1.00 | 1.000 | 0.000 | nan | nan |
| VAE | DBSCAN | 2.00 | 1.000 | 0.000 | nan | nan |
| DAGMM | KMeans | 0.10 | 0.838 | 0.053 | 0.566 | 0.553 |
| DAGMM | KMeans | 0.50 | 0.490 | 0.021 | 0.566 | 0.467 |
| DAGMM | KMeans | 1.00 | 0.321 | 0.009 | 0.566 | 0.403 |
| DAGMM | KMeans | 2.00 | 0.168 | 0.004 | 0.566 | 0.326 |
| DAGMM | FCM | 0.10 | 0.897 | 0.008 | 0.561 | 0.555 |
| DAGMM | FCM | 0.50 | 0.535 | 0.012 | 0.561 | 0.468 |
| DAGMM | FCM | 1.00 | 0.335 | 0.008 | 0.561 | 0.400 |
| DAGMM | FCM | 2.00 | 0.170 | 0.005 | 0.561 | 0.321 |
| DAGMM | DBSCAN | 0.10 | 0.767 | 0.028 | nan | nan |
| DAGMM | DBSCAN | 0.50 | 0.004 | 0.001 | nan | nan |
| DAGMM | DBSCAN | 1.00 | −0.012 | 0.001 | nan | nan |
| DAGMM | DBSCAN | 2.00 | −0.008 | 0.003 | nan | nan |
| OmniAnomaly | KMeans | 0.10 | 0.972 | 0.004 | 0.510 | 0.454 |
| OmniAnomaly | KMeans | 0.50 | 0.779 | 0.014 | 0.510 | 0.165 |
| OmniAnomaly | KMeans | 1.00 | 0.425 | 0.029 | 0.510 | 0.063 |
| OmniAnomaly | KMeans | 2.00 | 0.236 | 0.007 | 0.510 | 0.027 |
| OmniAnomaly | FCM | 0.10 | 0.952 | 0.004 | 0.501 | 0.438 |
| OmniAnomaly | FCM | 0.50 | 0.511 | 0.004 | 0.501 | 0.118 |
| OmniAnomaly | FCM | 1.00 | 0.164 | 0.002 | 0.501 | 0.118 |
| OmniAnomaly | FCM | 2.00 | 0.129 | 0.003 | 0.501 | 0.059 |

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| OmniAnomaly | DBSCAN | 0.10 | 1.000 | 0.000 | nan | nan |
| OmniAnomaly | DBSCAN | 0.50 | 1.000 | 0.000 | nan | nan |
| OmniAnomaly | DBSCAN | 1.00 | 1.000 | 0.000 | nan | nan |
| OmniAnomaly | DBSCAN | 2.00 | 1.000 | 0.000 | nan | nan |
| USAD | KMeans | 0.10 | 1.000 | 0.000 | 0.977 | 0.977 |
| USAD | KMeans | 0.50 | 1.000 | 0.000 | 0.977 | 0.976 |
| USAD | KMeans | 1.00 | 1.000 | 0.000 | 0.977 | 0.974 |
| USAD | KMeans | 2.00 | 1.000 | 0.000 | 0.977 | 0.968 |
| USAD | FCM | 0.10 | 0.940 | 0.005 | 0.478 | 0.475 |
| USAD | FCM | 0.50 | 0.734 | 0.007 | 0.478 | 0.445 |
| USAD | FCM | 1.00 | 0.531 | 0.009 | 0.478 | 0.411 |
| USAD | FCM | 2.00 | 0.297 | 0.007 | 0.478 | 0.373 |
| USAD | DBSCAN | 0.10 | 1.000 | 0.000 | nan | nan |
| USAD | DBSCAN | 0.50 | 0.995 | 0.010 | nan | nan |
| USAD | DBSCAN | 1.00 | 0.987 | 0.013 | nan | nan |
| USAD | DBSCAN | 2.00 | 0.952 | 0.017 | nan | nan |
| TranAD | KMeans | 0.10 | 1.000 | 0.000 | 0.998 | 0.998 |
| TranAD | KMeans | 0.50 | 1.000 | 0.000 | 0.998 | 0.997 |
| TranAD | KMeans | 1.00 | 0.987 | 0.020 | 0.998 | 0.997 |
| TranAD | KMeans | 2.00 | 0.974 | 0.021 | 0.998 | 0.996 |
| TranAD | FCM | 0.10 | 1.000 | 0.000 | 0.998 | 0.998 |
| TranAD | FCM | 0.50 | 1.000 | 0.000 | 0.998 | 0.997 |
| TranAD | FCM | 1.00 | 0.987 | 0.020 | 0.998 | 0.997 |
| TranAD | FCM | 2.00 | 0.978 | 0.022 | 0.998 | 0.996 |
| TranAD | DBSCAN | 0.10 | 0.980 | 0.019 | nan | nan |
| TranAD | DBSCAN | 0.50 | 0.965 | 0.017 | nan | nan |
| TranAD | DBSCAN | 1.00 | 0.960 | 0.012 | nan | nan |
| TranAD | DBSCAN | 2.00 | 0.959 | 0.026 | nan | nan |

**Table 8.3:** Full evaluation of clustering stability across all models at different noise levels SMAP D-1

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| PCA | KMeans | 0.10 | 0.998 | 0.001 | 0.549 | 0.548 |
| PCA | KMeans | 0.50 | 0.983 | 0.008 | 0.549 | 0.530 |
| PCA | KMeans | 1.00 | 0.886 | 0.142 | 0.549 | 0.489 |
| PCA | KMeans | 2.00 | 0.725 | 0.147 | 0.549 | 0.400 |
| PCA | FCM | 0.10 | 0.992 | 0.003 | 0.533 | 0.532 |
| PCA | FCM | 0.50 | 0.959 | 0.007 | 0.533 | 0.510 |
| PCA | FCM | 1.00 | 0.888 | 0.006 | 0.533 | 0.460 |
| PCA | FCM | 2.00 | 0.767 | 0.024 | 0.533 | 0.374 |
| PCA | DBSCAN | 0.10 | 0.961 | 0.011 | 0.358 | 0.365 |
| PCA | DBSCAN | 0.50 | 0.717 | 0.014 | 0.358 | 0.343 |
| PCA | DBSCAN | 1.00 | −0.002 | 0.001 | 0.358 | nan |
| PCA | DBSCAN | 2.00 | 0.000 | 0.000 | 0.358 | nan |

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| KPCA | KMeans | 0.10 | 0.998 | 0.002 | 0.558 | 0.558 |
| KPCA | KMeans | 0.50 | 0.976 | 0.007 | 0.558 | 0.549 |
| KPCA | KMeans | 1.00 | 0.942 | 0.006 | 0.558 | 0.526 |
| KPCA | KMeans | 2.00 | 0.814 | 0.017 | 0.558 | 0.451 |
| KPCA | FCM | 0.10 | 0.999 | 0.001 | 0.558 | 0.558 |
| KPCA | FCM | 0.50 | 0.982 | 0.004 | 0.558 | 0.548 |
| KPCA | FCM | 1.00 | 0.947 | 0.006 | 0.558 | 0.523 |
| KPCA | FCM | 2.00 | 0.710 | 0.020 | 0.558 | 0.437 |
| KPCA | DBSCAN | 0.10 | 0.991 | 0.004 | 0.316 | 0.320 |
| KPCA | DBSCAN | 0.50 | 0.637 | 0.004 | 0.316 | nan |
| KPCA | DBSCAN | 1.00 | 0.647 | 0.106 | 0.316 | nan |
| KPCA | DBSCAN | 2.00 | 0.000 | 0.000 | 0.316 | nan |
| RP | KMeans | 0.10 | 0.965 | 0.007 | 0.462 | 0.457 |
| RP | KMeans | 0.50 | 0.837 | 0.013 | 0.462 | 0.357 |
| RP | KMeans | 1.00 | 0.652 | 0.016 | 0.462 | 0.262 |
| RP | KMeans | 2.00 | 0.534 | 0.011 | 0.462 | 0.173 |
| RP | FCM | 0.10 | 0.957 | 0.006 | 0.435 | 0.424 |
| RP | FCM | 0.50 | 0.766 | 0.014 | 0.435 | 0.330 |
| RP | FCM | 1.00 | 0.633 | 0.012 | 0.435 | 0.258 |
| RP | FCM | 2.00 | 0.554 | 0.008 | 0.435 | 0.157 |
| RP | DBSCAN | 0.10 | 0.975 | 0.004 | 0.262 | 0.305 |
| RP | DBSCAN | 0.50 | 0.000 | 0.000 | 0.262 | nan |
| RP | DBSCAN | 1.00 | 0.000 | 0.000 | 0.262 | nan |
| RP | DBSCAN | 2.00 | 0.000 | 0.000 | 0.262 | nan |
| AE | KMeans | 0.10 | 0.994 | 0.002 | 0.498 | 0.496 |
| AE | KMeans | 0.50 | 0.968 | 0.008 | 0.498 | 0.460 |
| AE | KMeans | 1.00 | 0.917 | 0.008 | 0.498 | 0.389 |
| AE | KMeans | 2.00 | 0.727 | 0.023 | 0.498 | 0.265 |
| AE | FCM | 0.10 | 0.977 | 0.003 | 0.472 | 0.470 |
| AE | FCM | 0.50 | 0.876 | 0.011 | 0.472 | 0.434 |
| AE | FCM | 1.00 | 0.615 | 0.077 | 0.472 | 0.394 |
| AE | FCM | 2.00 | 0.570 | 0.037 | 0.472 | 0.279 |
| AE | DBSCAN | 0.10 | 0.964 | 0.011 | 0.375 | 0.363 |
| AE | DBSCAN | 0.50 | 0.458 | 0.017 | 0.375 | 0.019 |
| AE | DBSCAN | 1.00 | 0.007 | 0.004 | 0.375 | nan |
| AE | DBSCAN | 2.00 | 0.000 | 0.000 | 0.375 | nan |
| LSTM | KMeans | 0.10 | 0.984 | 0.007 | 0.590 | 0.589 |
| LSTM | KMeans | 0.50 | 0.938 | 0.007 | 0.590 | 0.584 |
| LSTM | KMeans | 1.00 | 0.863 | 0.015 | 0.590 | 0.575 |
| LSTM | KMeans | 2.00 | 0.720 | 0.018 | 0.590 | 0.559 |
| LSTM | FCM | 0.10 | 0.991 | 0.003 | 0.590 | 0.590 |
| LSTM | FCM | 0.50 | 0.945 | 0.007 | 0.590 | 0.585 |
| LSTM | FCM | 1.00 | 0.879 | 0.010 | 0.590 | 0.576 |
| LSTM | FCM | 2.00 | 0.737 | 0.014 | 0.590 | 0.558 |
| LSTM | DBSCAN | 0.10 | 0.975 | 0.006 | 0.343 | 0.375 |

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| LSTM | DBSCAN | 0.50 | 0.920 | 0.012 | 0.343 | 0.320 |
| LSTM | DBSCAN | 1.00 | 0.845 | 0.013 | 0.343 | 0.400 |
| LSTM | DBSCAN | 2.00 | 0.599 | 0.036 | 0.343 | 0.305 |
| TransAE | KMeans | 0.10 | 0.993 | 0.003 | 0.588 | 0.588 |
| TransAE | KMeans | 0.50 | 0.929 | 0.009 | 0.588 | 0.581 |
| TransAE | KMeans | 1.00 | 0.857 | 0.012 | 0.588 | 0.552 |
| TransAE | KMeans | 2.00 | 0.631 | 0.036 | 0.588 | 0.486 |
| TransAE | FCM | 0.10 | 0.988 | 0.004 | 0.588 | 0.588 |
| TransAE | FCM | 0.50 | 0.919 | 0.007 | 0.588 | 0.580 |
| TransAE | FCM | 1.00 | 0.844 | 0.009 | 0.588 | 0.552 |
| TransAE | FCM | 2.00 | 0.673 | 0.020 | 0.588 | 0.484 |
| TransAE | DBSCAN | 0.10 | 0.983 | 0.004 | 0.339 | 0.352 |
| TransAE | DBSCAN | 0.50 | 0.886 | 0.012 | 0.339 | 0.390 |
| TransAE | DBSCAN | 1.00 | 0.737 | 0.016 | 0.339 | 0.401 |
| TransAE | DBSCAN | 2.00 | 0.635 | 0.016 | 0.339 | 0.335 |
| VAE | KMeans | 0.10 | 0.979 | 0.005 | 0.569 | 0.559 |
| VAE | KMeans | 0.50 | 0.919 | 0.005 | 0.569 | 0.446 |
| VAE | KMeans | 1.00 | 0.811 | 0.016 | 0.569 | 0.342 |
| VAE | KMeans | 2.00 | 0.584 | 0.025 | 0.569 | 0.243 |
| VAE | FCM | 0.10 | 0.980 | 0.004 | 0.559 | 0.547 |
| VAE | FCM | 0.50 | 0.836 | 0.012 | 0.559 | 0.410 |
| VAE | FCM | 1.00 | 0.736 | 0.013 | 0.559 | 0.325 |
| VAE | FCM | 2.00 | 0.611 | 0.015 | 0.559 | 0.224 |
| VAE | DBSCAN | 0.10 | 0.911 | 0.011 | 0.338 | 0.282 |
| VAE | DBSCAN | 0.50 | 0.000 | 0.000 | 0.338 | nan |
| VAE | DBSCAN | 1.00 | 0.000 | 0.000 | 0.338 | nan |
| VAE | DBSCAN | 2.00 | 0.000 | 0.000 | 0.338 | nan |
| DAGMM | KMeans | 0.10 | 0.981 | 0.004 | 0.546 | 0.539 |
| DAGMM | KMeans | 0.50 | 0.878 | 0.014 | 0.546 | 0.445 |
| DAGMM | KMeans | 1.00 | 0.660 | 0.035 | 0.546 | 0.344 |
| DAGMM | KMeans | 2.00 | 0.411 | 0.013 | 0.546 | 0.254 |
| DAGMM | FCM | 0.10 | 0.981 | 0.004 | 0.543 | 0.534 |
| DAGMM | FCM | 0.50 | 0.751 | 0.015 | 0.543 | 0.409 |
| DAGMM | FCM | 1.00 | 0.587 | 0.015 | 0.543 | 0.334 |
| DAGMM | FCM | 2.00 | 0.411 | 0.011 | 0.543 | 0.202 |
| DAGMM | DBSCAN | 0.10 | 0.979 | 0.005 | 0.378 | 0.366 |
| DAGMM | DBSCAN | 0.50 | 0.566 | 0.010 | 0.378 | nan |
| DAGMM | DBSCAN | 1.00 | 0.006 | 0.004 | 0.378 | nan |
| DAGMM | DBSCAN | 2.00 | 0.000 | 0.000 | 0.378 | nan |
| OmniAnomaly | KMeans | 0.10 | 0.987 | 0.005 | 0.364 | 0.360 |
| OmniAnomaly | KMeans | 0.50 | 0.902 | 0.031 | 0.364 | 0.325 |
| OmniAnomaly | KMeans | 1.00 | 0.750 | 0.035 | 0.364 | 0.295 |
| OmniAnomaly | KMeans | 2.00 | 0.641 | 0.024 | 0.364 | 0.246 |
| OmniAnomaly | FCM | 0.10 | 0.978 | 0.004 | 0.356 | 0.352 |
| OmniAnomaly | FCM | 0.50 | 0.867 | 0.008 | 0.356 | 0.320 |

| Encoder | Clustering | $\sigma$ | ARI | ARI std | Sil. (clean) | Sil. (noisy) |
|---|---|---|---|---|---|---|
| OmniAnomaly | FCM | 1.00 | 0.764 | 0.010 | 0.356 | 0.287 |
| OmniAnomaly | FCM | 2.00 | 0.650 | 0.013 | 0.356 | 0.242 |
| OmniAnomaly | DBSCAN | 0.10 | 0.056 | 0.033 | nan | nan |
| OmniAnomaly | DBSCAN | 0.50 | 0.000 | 0.000 | nan | nan |
| OmniAnomaly | DBSCAN | 1.00 | 0.000 | 0.000 | nan | nan |
| OmniAnomaly | DBSCAN | 2.00 | 0.000 | 0.000 | nan | nan |
| USAD | KMeans | 0.10 | 0.976 | 0.005 | 0.627 | 0.624 |
| USAD | KMeans | 0.50 | 0.882 | 0.012 | 0.627 | 0.557 |
| USAD | KMeans | 1.00 | 0.751 | 0.015 | 0.627 | 0.469 |
| USAD | KMeans | 2.00 | 0.540 | 0.042 | 0.627 | 0.458 |
| USAD | FCM | 0.10 | 0.894 | 0.187 | 0.741 | 0.711 |
| USAD | FCM | 0.50 | 0.737 | 0.220 | 0.741 | 0.606 |
| USAD | FCM | 1.00 | 0.615 | 0.193 | 0.741 | 0.500 |
| USAD | FCM | 2.00 | 0.521 | 0.111 | 0.741 | 0.371 |
| USAD | DBSCAN | 0.10 | 0.917 | 0.008 | 0.455 | 0.442 |
| USAD | DBSCAN | 0.50 | 0.358 | 0.025 | 0.455 | −0.018 |
| USAD | DBSCAN | 1.00 | 0.010 | 0.006 | 0.455 | −0.268 |
| USAD | DBSCAN | 2.00 | 0.005 | 0.004 | 0.455 | nan |
| TranAD | KMeans | 0.10 | 1.000 | 0.000 | 0.846 | 0.844 |
| TranAD | KMeans | 0.50 | 1.000 | 0.000 | 0.846 | 0.818 |
| TranAD | KMeans | 1.00 | 1.000 | 0.000 | 0.846 | 0.766 |
| TranAD | KMeans | 2.00 | 0.858 | 0.003 | 0.846 | 0.590 |
| TranAD | FCM | 0.10 | 1.000 | 0.000 | 0.846 | 0.844 |
| TranAD | FCM | 0.50 | 0.575 | 0.000 | 0.846 | 0.466 |
| TranAD | FCM | 1.00 | 0.632 | 0.113 | 0.846 | 0.455 |
| TranAD | FCM | 2.00 | 0.659 | 0.129 | 0.846 | 0.403 |
| TranAD | DBSCAN | 0.10 | 0.925 | 0.014 | 0.381 | 0.296 |
| TranAD | DBSCAN | 0.50 | 0.000 | 0.000 | 0.381 | nan |
| TranAD | DBSCAN | 1.00 | 0.000 | 0.000 | 0.381 | nan |
| TranAD | DBSCAN | 2.00 | 0.000 | 0.000 | 0.381 | nan |

| Encoder | Space | Metric | Clustering | ARI | NMI | Silhouette |
|---|---|---|---|---|---|---|
| Raw | Raw | cosine | KMeans | 0.237 | 0.429 | 0.102 |
| PCA | Latent | cosine | DBSCAN | 0.486 | 0.608 | 0.116 |
| PCA | Latent+tSNE | euclidean | KMeans | 0.465 | 0.578 | 0.396 |
| PCA | Latent+UMAP | euclidean | KMeans | 0.572 | 0.679 | 0.582 |
| KPCA | Latent | cosine | KMeans | 0.075 | 0.289 | 0.726 |
| KPCA | Latent+tSNE | cosine | FCM | 0.310 | 0.386 | 0.449 |
| KPCA | Latent+UMAP | euclidean | FCM | 0.335 | 0.387 | 0.611 |
| RP | Latent | cosine | KMeans | 0.164 | 0.212 | 0.147 |
| RP | Latent+tSNE | euclidean | KMeans | 0.204 | 0.262 | 0.300 |
| RP | Latent+UMAP | cosine | KMeans | 0.248 | 0.311 | 0.504 |
| AE | Latent | cosine | KMeans | 0.418 | 0.501 | 0.218 |
| AE | Latent+tSNE | cosine | KMeans | 0.499 | 0.544 | 0.460 |
| AE | Latent+UMAP | euclidean | DBSCAN | 0.500 | 0.625 | 0.547 |
| VAE | Latent | cosine | KMeans | 0.359 | 0.441 | 0.235 |
| VAE | Latent+tSNE | cosine | KMeans | 0.412 | 0.470 | 0.429 |
| VAE | Latent+UMAP | euclidean | KMeans | 0.424 | 0.506 | 0.458 |
| LSTM | Latent | euclidean | KMeans | 0.349 | 0.460 | 0.289 |
| LSTM | Latent+tSNE | cosine | KMeans | 0.459 | 0.571 | 0.466 |
| LSTM | Latent+UMAP | euclidean | KMeans | 0.614 | 0.725 | 0.590 |
| TransAE | Latent | cosine | DBSCAN | 0.325 | 0.563 | 0.089 |
| TransAE | Latent+tSNE | euclidean | KMeans | 0.519 | 0.619 | 0.430 |
| TransAE | Latent+UMAP | euclidean | DBSCAN | 0.491 | 0.701 | 0.678 |
| USAD | Latent | cosine | KMeans | 0.225 | 0.351 | 0.227 |
| USAD | Latent+tSNE | cosine | FCM | 0.347 | 0.454 | 0.431 |
| USAD | Latent+UMAP | euclidean | DBSCAN | 0.309 | 0.514 | 0.301 |
| DAGMM | Latent | cosine | KMeans | 0.306 | 0.356 | 0.149 |
| DAGMM | Latent+tSNE | euclidean | KMeans | 0.325 | 0.391 | 0.371 |
| DAGMM | Latent+UMAP | euclidean | KMeans | 0.342 | 0.400 | 0.467 |
| OmniAnomaly | Latent | cosine | KMeans | 0.298 | 0.388 | 0.130 |
| OmniAnomaly | Latent+tSNE | cosine | KMeans | 0.239 | 0.330 | 0.372 |
| OmniAnomaly | Latent+UMAP | cosine | FCM | 0.199 | 0.328 | 0.476 |

**Table 8.4:** Full clustering results across encoders, spaces, metrics, and clustering methods.