ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Artificial Intelligence in Industry

ANOMALY DETECTION IN RAILWAY RADIO SIGNALING: SYNTHETIC DATA GENERATION AND AI-BASED MODELS

CANDIDATE

Riccardo Romeo

SUPERVISOR

Prof. Andrea Borghesi

CO-SUPERVISOR

Prof. Paolo Torroni

Academic year 2024-2025

Session 4th

Abstract

The increasing reliance on radio-based communication in modern railway systems, combined with stringent safety requirements, makes the design and validation of Wi-Fi networks a critical and complex task. In the context of the Torino Linea 1 metro line, even minor anomalies in radio signal performance can significantly impact operational safety and network reliability.

This thesis presents the development of an automated *Radio Network Anomaly Detection* tool designed to assist network designers during the early stages of Wi-Fi planning. The tool leverages an ensemble of Artificial Intelligence techniques, including Autoencoder, Gaussian Mixture Models, and Isolation Forest, to identify anomalies in radio signal.

A key contribution of this work is the creation of a synthetic data generation pipeline. Multiple models were extensively tested to determine the most effective method for producing high-quality, anomaly-free data, which is scarce in early design phases. The combination of a Tabular Variational Autoencoder (TVAE) with a rejection sampling strategy, guided by a Random Forest discriminator, emerged as the most effective pipeline, significantly enhancing the performance of anomaly detection models.

The experiments show that the Autoencoder model achieves the highest performances. Overall, the developed tool provides network designers with a reliable solution for evaluating radio communication performance, reducing design time, minimizing costly rework, and ultimately contributing to the construction of safer railway infrastructures.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, the increasing level of automation in railway and metropolitan transportation systems has led to a growing dependence on advanced communication infrastructures. Modern signaling architectures rely heavily on continuous and reliable data exchange between trains and trackside equipment to ensure safe and efficient operations. In this context, radio communication systems have become a critical component, particularly in driverless and semi-automatic metro lines, where the absence of human operators further amplifies the need for robust and resilient communication networks. Consequently, the design, monitoring, and validation of such radio networks represent a key challenge for both infrastructure managers and system integrators.

Within this framework, the Torino Linea 1 metro system represents a highly complex case study. The extensive use of radio-based communication, coupled with the stringent safety and availability requirements of Communication Based Train Control (CBTC) systems, makes designing the Wi-Fi infrastructure a critical and non-trivial task. Even minor anomalies in radio performance, such as interference, insufficient coverage, or congestion, can significantly affect system reliability and operational safety. Thus, analyzing these signals is essential, yet manual inspection is time-consuming, labor-intensive, and prone to human errors. These challenges underscore the need for automated approaches to support network planning and ensure accurate assessment of radio performance.

Motivated by these considerations, the primary objective of this thesis is the development of an automated Radio Network Anomaly Detection tool. The tool is designed to assist network designers during the early stages of Wi-Fi network planning by automatically identifying anomalies and criticalities in radio signal performance. A particular challenge in these early stages is the limited availability of anomaly-free data, which can hinder accurate model training. To address this limitation, a pipeline for generating synthetic data using Artificial Intelligence models has been implemented.

Overall, the results demonstrate that the novel synthetic data generation pipeline developed in this thesis, based on a Tabular Variational Autoencoder coupled with a rejection sampling mechanism guided by a Random Forest classifier, successfully produces synthetic data highly similar to real, anomaly-free measurements. This approach effectively addresses the scarcity of clean training data in the early stages of network design, enhancing the performance of the anomaly detection models considered in this work. Among these models, the Autoencoder consistently exhibits the strongest capability to identify anomalies, outperforming Gaussian Mixture Models and Isolation Forest in both sensitivity and robustness.

The structure of this thesis is organized as follows:

- **Chapter 2: Background**: This chapter provides the background information on the Turin Metro Line 1 and the operational principles of the CBTC protocol. It also introduces the theoretical foundations of all AI-based models employed in this work, including those used for anomaly detection and synthetic data generation, as well as the metrics applied for model validation.

- **Chapter 3: Related Works**: This chapter presents a brief review of previous studies related to anomaly detection in railway applications, alongside a summary of relevant works on synthetic data generation.

- **Chapter 4: Data** This chapter describes the dataset used in this work, detailing both the data collection process conducted in the field and the expert labelling performed subsequently.
- **Chapter 5: Method**: This chapter outlines the methodology adopted for both the synthetic data generation and the anomaly detection tasks.
- **Chapter 6: Experimental Results**: This chapter reports the results of the thesis, including outcomes from both the anomaly detection models and the synthetic data generation methods.
- **Chapter 7: Conclusions**: This chapter summarizes the key findings of the thesis, discusses the impact of the work within the considered scenario, and provides a critical analysis along with suggestions for potential future improvements.

Chapter 2

Background

2.1 Railway and Radio Infrastructure

The **“Torino Linea 1” metro line**, comprising a total of 27 stations, represents an extremely ambitious project from the perspective of automation. The entire infrastructure is designed to be almost entirely automated, and signaling plays a crucial role in achieving this goal. The map of the infrastructure is shown in Figure 2.1. The project is currently held by **Alstom Ferroviaria s.p.a.**

Unlike other railway infrastructures, in this metro line, most of the signaling is delegated to the train itself, rather than relying on a control center tasked with coordinating the entire rail traffic along the line. Here, it is the trains themselves that communicate, through the infrastructure, the use or non-use of infrastructural resources (such as switches) to other trains. The entire communication takes place entirely through wired and wireless systems.

The type of communication system used enables the construction of a driverless or semi-automatic infrastructure, allowing for continuous and dynamic control of trains in operation.

In particular, this communication occurs, in part, through radio-based train-to-ground and ground-to-train signaling. Therefore, the communication between trains is not direct but takes place with the assistance of infrastructural equipment. In the following discussion, we will refer to this concept using the

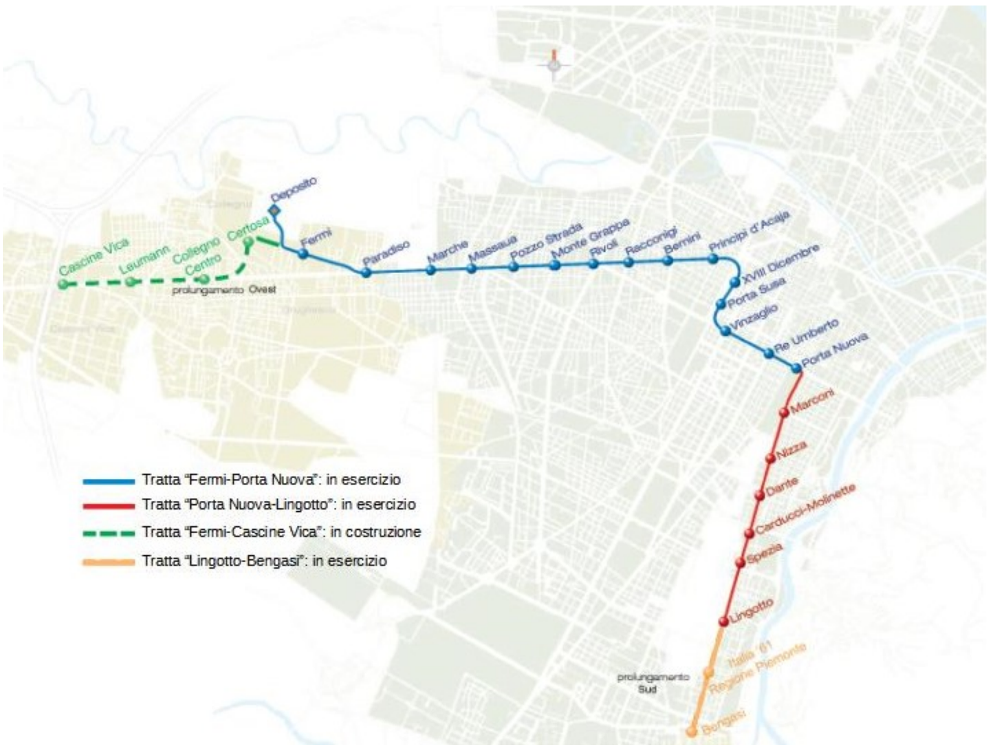


Figure 2.1: Map of Turin Metro Line 1. The blue section represents the Fermi–Porta Nuova segment, the red section indicates the Porta Nuova–Lingotto segment, the green dashed section corresponds to the planned Fermi–Cascine Vica extension, and the yellow section shows the Lingotto–Bengasi segment. Image source: Infra.to.

term **”backbone”**.

The infrastructure features two main backbones: one dedicated to **Communication Based Train Control (CBTC)** system, while the other facilitates the transmission of multimedia data. CBTC is a communication standard for railway infrastructures proposed by IEEE Association, this standard establishes a set of functional requirements necessary for enhancing performance, availability, operations, and train protection [21]. Both backbones are designed with a ”ring” topology.

The first of these backbones consists of a dual ring: two independent networks called **RED** and **BLUE**. All the information necessary for the operation of the CBTC system is transmitted by signaling devices on both networks simultaneously. ”Twin” packets travel along two separate paths. This redundancy is designed for safety reasons: the failure of one of the two networks (BLUE or RED) has no impact on the operation of the second network, nor, therefore, on the functioning of the system as a whole.

The second backbone is composed by only one network called **GREEN**, used entirely for multimedia data. This stream is, for instance, used to transmit all the images from security cameras installed on train board.

Radio Infrastructure The ground-to-train communication is carried out by a system of WiFi Radio Access Points (APs) distributed along the line. There are three physically independent Access Point systems for the RED, BLUE (CBTC), and MULTIMEDIA (Multimedia Services) networks. They provide continuous coverage along the entire line.

Of course, the wireless communication system is then connected to the wired backbones, maintaining data separation.

The radio communication is therefore both continuous and mobile. To ensure continuous communication, the onboard Radio device, called the Onboard Radio Modem (OBM), must always be connected to a Radio Access Point along the line.

Mobile Communication The communication is made mobile precisely thanks to the radio infrastructure.

When a train reaches the coverage limit of a Radio Access Point, the quality of the radio signal decreases. In this case, the onboard radio modems (BLUE, RED, and MULTIMEDIA, also called GREEN) must perform what is known as a hand-off, i.e., they must find a radio signal of better quality to ensure high-quality communication between the train and the ground infrastructure. The hand-off is initiated by the onboard radio modem and occurs in three phases:

- Detection;
- Scanning;
- Association.

Detection The Onboard Radio Modem is responsible for continuously monitoring the quality of the existing radio communication. This quality is checked by measuring the signal strength.

Scanning If the quality of the radio signal falls below a predefined threshold, the radio modem enters the mode of scanning existing radio networks.

During this phase, the radio modem continues to maintain communication with the previous Radio Access Point while simultaneously searching for a higher-quality communication link.

Association When the train's radio modem finds a new Radio Access Point that better meets the criteria for radio link quality, it enters the association phase, i.e., it establishes a new physical connection that will carry the ground-to-train communications.

During the association phase, the train's radio modem performs authentication and association with the new Radio Access Point and sends a release message to the current Radio Access Point. The new Radio Access Point (on the ground) receives an authentication request and then an association request from the train's radio modem. After association, the train's radio modem sends a packet to inform the wired transport network of the train's new location on the network.

Trackside Radio Equipment Trackside Radio Equipment (TRE) refers to the set of devices useful for radio communication on the trackside, such as antennas, cables, and WiFi radio modems.

On the ground, each Radio Access Point is connected to a series of directional antennas, each capable of covering a signal propagation direction to ensure the best possible coverage.

Along the line, the Radio Access Points for each of the backbones (BLUE, RED, and GREEN) are enclosed and protected by what is called a TRE-BOX.

Onboard Radio Equipment Onboard the train, there are two Onboard Radio Modems for CBTC communication and two for multimedia communication. Thus, one OBM is dedicated to the RED line, one to the BLUE line, and two to the GREEN line.

In particular, these Onboard Radio Modems are located at the front or rear of the train in the cabins (Figure 2.2).

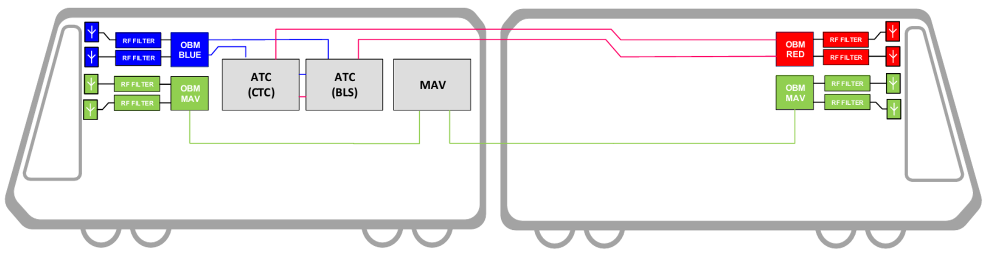


Figure 2.2: On-Board Radio Modem Architecture

2.2 Anomaly Detection

Based on IBM article: [24], Anomaly Detection, or Outlier Detection, is defined as:

> *The identification of observations, events or data points that deviate from what is usual, standard or expected, making them inconsistent with the rest of a data set.*

Nowadays, anomaly detection is a process that leverages the use of machine learning and deep learning techniques. Anomalies in data can potentially identify incidents, errors, or even infrastructural failures. These anomalies can be complex to identify, even for domain experts. Additionally, in most real-world applications, outliers appear as extremely rare cases.

2.2.1 Anomaly Detection in Railway Radio Signaling

In the case study presented in this work, the primary objective was to identify anomalies occurring in the radio communication signals along the railway line. Detecting such anomalies is a key aspect in the design, optimization, and maintenance of the railway communication infrastructure. Ideally, if the system operates correctly, the radio signals should consistently follow a stable and characteristic distribution. Therefore, deviations from this expected behaviour may indicate potential malfunctions or inefficiencies within the infrastructure.

For this project, we collaborated closely with domain experts, specifically telecommunications engineers responsible for designing and testing the railway radio network. According to their expertise, the anomalies detected within the system can be classified into two main categories:

- **Deterministic Anomalies**: anomalies that can be detected through direct and rule-based analysis of the data, without requiring subjective interpretation.

- **Signal Anomalies**: anomalies that can only be identified through expert assessment, by visually inspecting the distributions and patterns of the radio signals.

Signal anomalies generally indicate disturbances in the radio communication within specific areas of the network. Such disturbances may arise from several factors, including signal reflections along the tunnel walls, excessive proximity between adjacent APs, electromagnetic shielding caused by passing trains, or malfunctions in the APs or their antennas.

In the railway system considered, failures in a single backbone (either RED or BLUE) do not cause the infrastructure to shut down. However, if both backbones fail simultaneously, the safety mechanisms of the system are automatically triggered, leading to an emergency braking procedure. Consequently, the early identification of these anomalies is fundamental to ensuring that the infrastructure operates efficiently and safely, while maintaining continuous and reliable radio communication. Detecting anomalies allows telecommunications engineers to trace the root causes of malfunctions during the design or testing phases and to apply corrective measures.

Under normal operating conditions, communication between the train and the ground network follows a predictable sequence: the OBM connects sequentially to each Radio Access Point along the backbone as the train moves. Therefore, by analysing the order of these connections, it is possible to detect deterministic anomalies and undesired behaviours automatically. The domain experts identified three main types of deterministic anomalies: the *radio jump*, the *self-handoff*, and the *color change*.

Specifically, a *radio jump* occurs when the OBM connects non-sequentially to the APs, skipping one or more of them. A *self-handoff* takes place when the OBM disconnects from a AP and immediately reconnects to the same one. Finally, in the case of the CBTC backbones (RED and BLUE), a *color change* anomaly occurs when the OBM connected to one backbone temporarily connects to an access point belonging to the other backbone.

In ideal conditions, no deterministic or signal anomalies should be present. While deterministic anomalies can be automatically detected using algorithmic or rule-based approaches, signal anomalies are more complex and typically require expert knowledge to be identified. However, it is possible to develop AI-based systems capable of learning from historical data to automatically recognize and classify such signal anomalies, thus supporting the experts in deployment activities.

2.2.2 Anomaly Detection Methods

The methods for anomaly detection based on Artificial Intelligence techniques are numerous. Below, we present the methods we used in this project.

To identify anomalies in our use case, we decided to use the following three models for anomaly detection: **Isolation Forest (IF)**, **Gaussian Mixture Models (GMM)**, and **Auto-Encoder (AE)**.

2.2.2.1 Isolation Forest

Isolation Forest, also known as iForest or IF, is a model based on the idea that anomalies in data can be identified without previously training a model on data with zero anomalies.

As stated in the original paper introducing Isolation Forest [30]:

> *This is a model-based method that explicitly isolates anomalies rather than profiles normal instances.*

Additionally, the paper specifies that this method leverages the fact that anomalies are fewer in number compared to the rest of the data and that they have attribute values significantly different from other points.

The entire method is based on what the authors call an **Isolation Tree**. This is essentially a binary tree in which, given a node T, it could be an external node without children or an internal node with two daughter nodes and a test.

A test consists of an attribute and a split value, which divides the attribute into the first or second daughter node.

Thus, given a sample of data $X = \{x_1, ..., x_n\}$ with n instances, this sample is recursively divided using randomly selected attributes and split values. It is clear that an instance is isolated to an external node when an Isolation Tree is fully grown, in which case the number of external nodes is n.

To identify anomalies in samples, the authors propose an anomaly metric based on the path length of points. The general idea is that if the path length is short, it means the point has been isolated earlier compared to others, making it more likely to be an anomaly.

Note that the entire process is based on building what is called a **forest**, which is a set of Isolation Trees. If a forest produces shorter path lengths for some samples, these samples are more likely to be anomalies. This process is shown in Figure 2.3.

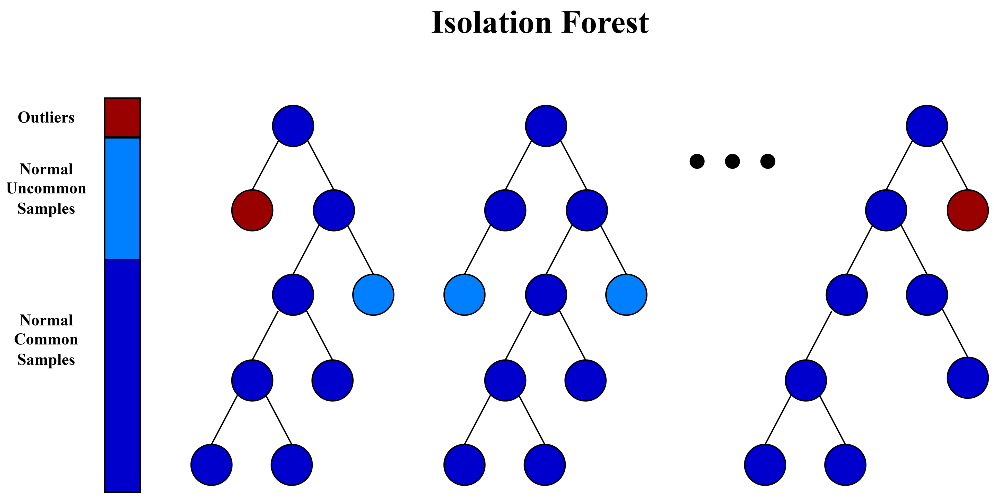


Figure 2.3: Isolation Forest algorithm.

2.2.2.2 Gaussian Mixture Models

Gaussian Mixture Models (GMM) are probabilistic models that represent data distributions as a combination of two or more Gaussian (normal) distributions.

As stated in Chapter 6 of the book *Algorithmic Aspects of Machine Learning* [35]:

> *A mixture of two Gaussians is a distribution whose density function is:*
>
> $$F(x) = w_1 F_1(x) + (1 - w_1) F_2(x)$$
>
> *where F_1 and F_2 are Gaussians. We can generate a random sample as follows: with probability w_1, we output a random sample from F_1, and otherwise, we output a random sample from F_2.*

This concept generalizes to n Gaussian components, allowing flexible modeling of complex distributions. However, the method relies on the assumption that data can be approximated by Gaussian components—an assumption that may not always hold.

For a multivariate Gaussian distribution, the probability density function is:

$$\mathcal{N}(\mu, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{n/2} \det(\mathbf{\Sigma})^{1/2}} \exp\left(-\frac{(\mathbf{x} - \mu)^\top \mathbf{\Sigma}^{-1} (\mathbf{x} - \mu)}{2}\right)$$

Here, $\mathbf{\Sigma}$ is the covariance matrix, and μ is the mean vector.

To fit a GMM to data, the parameters (means, covariances, and mixture weights) must be optimized. Ideally, it is possible to optimize these parameters using maximum likelihood estimator, ”*which would choose to set the parameters to as to maximize the probability that the mixture would generate the observed samples*” [35]. But, this method presents an important limitation: it is a NP-hard problem. This is why today the standard approach uses the *expectation-maximization (EM)* algorithm, which iteratively refines the parameters to maximize the likelihood of the observed data. The main problem of this algorithm is that it could be stuck in a local optima, not finding the absolute one, and also, it is extremely sensitive on how it is initialized.

Gaussian Mixture Models can be used in order to perform Anomaly Detection tasks. In order to do that, the GMM model is fit with only non-anomalous data. Then, it is possible, for each other data point, to compute the probability density using the parameters learned during training. Thus, a metric that specifies how much a data sample is anomalous or not is required. In order to do that, usually, the negative log-likelihood of the probability density is used:

$$\text{Anomaly Score}(x) = -\log p(x)$$

A high anomaly score indicates that the data point is unlikely to belong to the learned distribution and is thus considered an anomaly. On the other hand, a low anomaly score value could means that the data point fits well within the learned distribution and can be considered normal.

2.2.2.3 Autoencoder

An **Autoencoder** can be defined as a neural network whose primary purpose is to learn an informative representation of data by reconstructing a set of input observations [34]. Specifically, the learning process of autoencoders is unsupervised [3].

Typically, an autoencoder consists of three main components:

- A neural network called the **encoder**.
- A **latent feature representation** (also referred to as the bottleneck).
- Another neural network called the **decoder**.

The encoder and decoder are functions, while the latent space representation is usually a tensor of real numbers [34]. The general architecture of an autoencoder is shown in Figure 2.4.

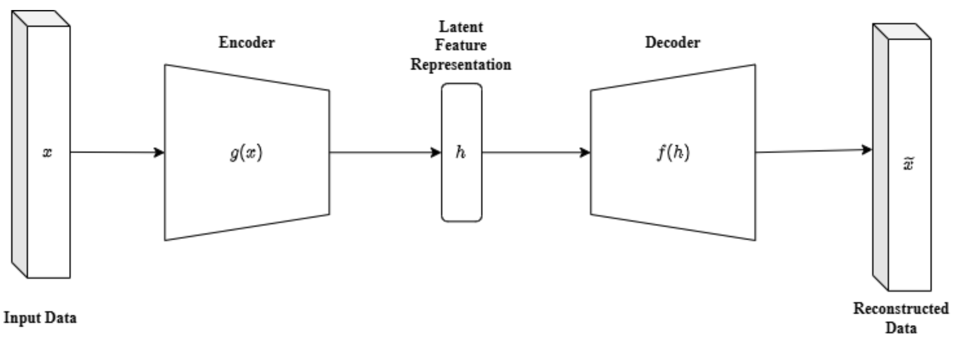


Figure 2.4: Classic Autoencoder architecture consisting of an encoder network g that compresses the input data x into a latent representation h, and a decoder network f that outputs the reconstructed input $\tilde{x}$. The model is trained to minimize the reconstruction error between the input and the output.

The encoder can be represented as a function g that produces the latent feature representation h. Given a sample of input data $X = \{x_1, \ldots, x_n\}$ of size n, we can express this as:

$$h_i = g(x_i),$$

where $h_i \in \mathbb{R}^q$, and thus $g : \mathbb{R}^n \to \mathbb{R}^q$. Similarly, the decoder can be represented as a function f that takes the latent feature representation h_i and produces the output $\tilde{x}_i$:

$$\tilde{x}_i = f(h_i) = f(g(x_i)).$$

Here, $\tilde{x}_i$ has the same dimensionality as the input x_i, i.e., $\tilde{x}_i \in \mathbb{R}^n$.

The training procedure of an autoencoder involves optimizing the encoder and decoder functions such that:

$$\arg\min_{f,g} \langle \Delta(x_i, \tilde{x}_i) \rangle ,$$

where Δ is a distance measure, and $\langle \cdot \rangle$ denotes the average over all observations in the sample. This is referred to as the **Loss Function**, which is minimized to set the parameters of the encoder and decoder. A commonly used loss function for autoencoders is the Mean Squared Error (MSE) [34].

The Mean Squared Error (MSE) is defined as:

$$\text{MSE} = \frac{1}{M}\sum_{i=1}^{M} |x_i - \widetilde{x}_i|^2,$$

where $|\cdot|$ represents the norm of a vector, and M is the number of observations in the training dataset. The absolute minimum of this function is achieved when $\widetilde{x}_i = x_i$.

During the unsupervised representation learning, the autoencoder adapts its parameters θ and θ' to minimize the loss function, which represents a difference metric between training samples x_i and their corresponding reconstructions $\widetilde{x}_i$, i.e.,

$$\theta^*, \theta'^* = \arg\min_{\theta,\theta'} \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\Big(x_i, \widetilde{x}_i\Big) = \arg\min_{\theta,\theta'} \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\Big(x_i, f_\theta(g_{\theta'}(x_i))\Big),$$

where f_θ and $g_{\theta'}$ denote the encoder and decoder functions parameterized by θ and θ', respectively, and $\mathcal{L}$ is the loss function.

Another crucial concept in the context of autoencoders is the **Reconstruction Error (RE)**. This function measures how closely the reconstructed value $\widetilde{x}_i$ matches the original input x_i. In other words, it quantifies the autoencoder's ability to reconstruct input data accurately. In anomaly detection, the reconstruction error is used to identify whether a given sample is anomalous. By training the autoencoder exclusively on non-anomalous data, it becomes adept at reconstructing such data. Consequently, when an anomalous data point is input, the autoencoder fails to reconstruct it effectively, resulting in an increased reconstruction error.

Commonly used reconstruction error metrics include the Mean Squared Error (MSE), the Sum of Squared Errors (SSE), and the Mean Absolute Error (MAE).

The Sum of Squared Errors (SSE) is defined as:

$$\text{SSE} = \sum_{i=1}^{M} |x_i - \widetilde{x}_i|^2,$$

where M retains its previous meaning. A limitation of this metric is that it tends to produce large values when the number of features in the dataset is high, due to the squaring operation. This is one reason why the Mean Absolute Error (MAE) is sometimes preferred.

The Mean Absolute Error (MAE) is expressed as:

$$\text{MAE} = \frac{1}{M} \sum_{i=1}^{M} |x_i - \widetilde{x}_i|,$$

where M is as defined earlier. This metric generally yields smaller values for the reconstruction error computed on anomalous data.

2.2.3 Thresholding Methods

As observed, several anomaly detection methods generate a signal as their output. In the context of **Autoencoders**, this signal is represented by the **Reconstruction Error**, which is computed for each input sample fed into the model. Similarly, **Gaussian Mixture Models** (GMMs) produce a **probability density signal** as their output.

This implies the necessity of a threshold to determine whether a specific data point constitutes an anomaly. Specifically, if the signal exceeds this threshold, the point is classified as an anomaly; otherwise, it is considered normal. Consequently, the selection of an appropriate **Threshold Optimization Method** (TOM) becomes crucial.

In the literature, there is a substantial body of research focused on testing and enhancing anomaly detection methods. However, comprehensive studies on thresholding methods are relatively scarce. Typically, thresholding methods can be categorized into two main families: **supervised** and **unsupervised** approaches [27].

In the following sections, we will present the **unsupervised thresholding methods** employed in this work.

2.2.3.1 Percentile TOM

The concept behind this first method is straightforward. Given the anomaly scores, the threshold is set at the k-th percentile of these scores. Specifically, the k-th percentile is the value below which k percent of the data falls.

It is important to note that this method is parametric. Indeed, it requires the specification of the parameter k, and the search for this parameter represents the primary limitation of the method.

2.2.3.2 Z-Score TOM

This second unsupervised algorithm is based on statistical assumptions. In particular, it assumes that the number of anomalies in the data is extremely small, as is often the case. Given the anomaly score values, we infer that those associated with anomalies (i.e., higher values) should lie far from the mean.

Therefore, we use a measure of how far a value deviates from the mean. To achieve this, we apply the Z-Score evaluation, which quantifies the distance of a point from the mean.

In particular, we have that the Z-Score can be computed as:

$$Z = \frac{X - \mu}{\sigma}$$

Where X is the value of the data point, μ is the mean and σ is the standard deviation.

Note that, a Z-Score equal to 0 means that the data point is exactly at the mean. While a Z-Score higher than 0 means that the data point has a value higher respect to the mean, and a Z-Score less than 0 means that the data point has a value below the mean.

In literature a value of Z-Score equal to 3 or -3 is used to detect anomalies [62]. Thus, if an anomaly score has a Z-Score higher than 3 we can consider it as an anomaly. The threshold value is selected as the minimum anomaly score that has a Z-Score greater than 3.

2.2.3.3 K-Means TOM

As reported by Komadina et al. [27], this thresholding method was introduced by Wang et al. [63] to handle data that do not follow a Gaussian distribution and contain a very small number of anomalies, specifically in the domain of railway transportation. The initial step of this method relies on the K-Means clustering algorithm.

The algorithm begins by computing an initial threshold using K-Means clustering. In particular, the number of clusters is set to three: one cluster for all non-anomalous points, another for points considered as transitional, and a third cluster for the anomalous points. The initial cluster centers are set as follows: 0, the mean of the anomaly scores, and the maximum anomaly score, respectively. The initial threshold is then chosen as the minimum value within the third cluster. Using a predefined granularity value, a set of candidate thresholds is generated.

For each candidate threshold, a ”goodness” metric S is computed, where a higher value indicates a better threshold:

$$S = \frac{\sigma(X)}{\sigma(X_l) \times |X_h|}$$

The terms in the formula are defined as follows:

- X_l: set of all anomaly values smaller than or equal to the threshold;
- X_h: set of all anomaly values greater than the threshold;
- $\sigma(X)$: standard deviation of all anomaly scores;

- $\sigma(X_l)$: standard deviation of all anomaly scores smaller than or equal to the threshold;

- $|X_h|$: number of elements in the set X_h.

The threshold corresponding to the highest score S is selected as the final threshold.

It is important to note that this algorithm is entirely non-parametric, as it does not require any additional parameters. However, the score S can be considered a heuristic, and thus the theoretical foundation behind it is not very strong. Moreover, since this method relies on a heuristic, its sensitivity to the type of data can be high, potentially limiting its adaptability to different datasets.

2.2.3.4 DBSCAN TOM

One limitation of the previous thresholding technique is the assumption that the data can be divided into three clusters. In practice, this means that prior knowledge about the data is required. To address this issue, Komadina et al. [27] proposed a method based on the **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)**[10] algorithm. As stated in their paper:

> *In density-based clustering, clusters are detected as regions where the objects are densely packed. The clusters are separated from each other by regions of low density.*

DBSCAN is thus used to assign labels to all anomaly scores. The authors base their algorithm on the number of classes detected by DBSCAN:

- If DBSCAN finds only one class, it means that anomalous values cannot be identified;

- If DBSCAN finds two classes, this represents the nominal case where it is possible to distinguish between anomalous and non-anomalous data;

- If DBSCAN finds three classes, it is not possible to select a threshold, and the result is considered undefined.

Furthermore, the authors specify that in the first case the threshold is set as the sum of the maximum and mean values of the anomaly scores. In the second case, the threshold is computed as the midpoint between the boundaries of the two groups identified by DBSCAN. In the third case, since it is considered undefined, the threshold is set to zero.

The main limitation of this method is its assumption that the groups of anomaly scores and non-anomalous scores are well separated, which may not always hold true. Additionally, the method depends on two parameters: *eps*, the radius defining cluster neighborhoods in DBSCAN, and *minPts*, the minimum number of points required to form a cluster.

2.2.3.5 Difference Between The Sets TOM

This unsupervised thresholding method, called **Difference Between The Sets (DBS)**, is entirely based on the assumption that the anomaly scores can be divided into a high-density region corresponding to non-anomalous data, and a low-density region corresponding to anomalous data. As reported by Komadina et al. [27], this method was first introduced by He et al. [19]. The core idea is that the threshold can be defined as the value that maximizes the distance between these two sets of anomaly scores.

This method relies on a metric representing the distance between the two sets:

$$d(X_l, X_h) = \frac{\min(X_h) - \max(X_l)}{\min(X_h) + \max(X_l) - 2 \times \min(X_l)}$$

where:

- X_l represents the set of anomaly scores smaller than or equal to the threshold;

- X_h represents the set of anomaly scores greater than the threshold.

The algorithm begins by setting a lower and an upper bound for the candidate thresholds. The lower bound, which also serves as the initial threshold, is set to the average of the anomaly scores, while the upper bound is set to their maximum value. Then, given a certain granularity, a list of candidate thresholds is generated. For each candidate threshold, the two sets X_l and X_h are computed, and the distance between them is evaluated. The final threshold is chosen as the one associated with the maximum distance.

Again, the main limitation of this algorithm lies in its fundamental assumption. If no anomalous data points are present in the dataset, the algorithm is not suitable for the situation.

2.2.3.6 Standard Deviations Inflection Point TOM

Standard Deviations Inflection Point (SDIP) is an unsupervised thresholding method proposed by Komadina et al. [27]. The idea behind this method is that, given the anomaly scores, it is possible to measure how much each value deviates from the others by computing the standard deviation of the set and iteratively removing the largest value. In the first iterations, the standard deviation typically decreases sharply, then gradually the decrease flattens. The point at which the rate of decrease changes can be used as a threshold because it indicates the moment when values being removed are no longer associated with real anomalies.

First, the algorithm sorts the anomaly scores in decreasing order. To speed up the process, only a certain percentage of these scores is used to compute the standard deviation. Then, iteratively, the largest value is removed, and the standard deviation is recalculated.

At this stage, all the standard deviation values can be plotted, and the inflection point can be identified using a Knee Point detection algorithm.

A limitation of this thresholding method is its dependence on the Knee locator algorithm, which may not be effective in all cases. Moreover, if multiple knees are present, the algorithm might detect the first one, potentially getting stuck in a local optimum.

2.2.4 Methods Evaluation

In the field of anomaly detection, and more broadly in AI-based systems, it is essential to evaluate the developed models to determine whether they effectively solve the intended task. In other words, it is necessary to rely on quantitative metrics, also referred to as *Key Performance Indicators (KPIs)*, which provide measurable evidence of the performance and success of an artificial intelligence system.

Several types of KPIs exist: some are designed for specific applications, while others are recognized as state-of-the-art metrics within certain domains. In the context of anomaly detection, it is particularly important to assess how many anomalies are correctly identified by the system and how many are missed. In most cases, a trade-off must be achieved between the number of missed anomalies and false detections. Ideally, a system that never misses any anomaly would be optimal; however, in practice, models that detect more anomalies often tend to generate a higher number of false positives, that is, they classify normal instances as anomalies. Therefore, balancing these two aspects is crucial, and KPIs serve as fundamental tools for determining which model performs better in relative terms.

In the following subsections, several metrics commonly used to evaluate anomaly detection systems are presented and discussed.

2.2.4.1 Confusion Matrix

A **Confusion Matrix** is a tabular representation used to evaluate the performance of a classification model by comparing the actual outcomes with the predicted ones. It serves as a visualization technique for understanding the results produced by a classifier [37]. It is important to recognize that, fundamentally, an anomaly detection task can be viewed as a classification problem, where the model must determine whether a record is anomalous or normal. In certain cases, the method must also identify the *type* of anomaly, thus addressing a multiclass classification problem. This is why the confusion matrix is a standard analytical tool in this field.

Typically, the columns of the confusion matrix represent the predicted classes, while the rows represent the actual classes (or vice versa). In the case of a binary classifier, the matrix can be interpreted as follows: the top-left cell corresponds to the number of *true positives (TP)*, representing cases where the model correctly predicts the positive class. Directly below it, the *false positives (FP)* represent instances from the negative class that were incorrectly classified as positive, in statistical terms, these are known as type I errors. The top-right cell contains the *false negatives (FN)*, which are actual positive instances that the model incorrectly labels as negative. Finally, the bottom-right cell reports the *true negatives (TN)*, denoting negative-class instances correctly identified as such. Summing all four values yields the total number of predictions made by the model.

In anomaly detection, TP represents the number of anomalies correctly identified, FP the number of normal records incorrectly classified as anomalies, TN the number of correctly classified normal instances, and FN the number of missed anomalies. Particular attention is often given to the number of TP, as in most applications it is preferable to employ a system that detects the highest possible number of anomalies.

2.2.4.2 Conditional Measures

Conditional Measures are metrics that describe a model's accuracy in identifying instances of a specific class or non-class. Among the most widely used are *Recall*, also known as the *True Positive Rate (TPR)* or *Sensitivity*, which measures the proportion of correctly predicted positive instances among all actual positives, and the *True Negative Rate (TNR)*, also referred to as *Specificity*, which represents the proportion of correctly predicted negatives among all actual negatives.

Formally, Sensitivity is computed as:

$$\text{Recall} = \text{TPR} = \frac{TP}{TP + FN}.$$

Whereas Specificity is defined as:

$$\text{TNR} = \frac{TN}{TN + FP}.$$

Another important conditional measure in anomaly detection is the *False Negative Rate (FNR)*, which quantifies the proportion of positive cases incorrectly predicted as negative, in other words, the proportion of anomalies that the model fails to detect. It can be computed as:

$$\text{FNR} = \frac{FN}{TP + FN}.$$

This metric, also known as the *Miss Rate*, is the complement of Sensitivity.

2.2.4.3 Unconditional Metrics

Unconditional Metrics represent the likelihood that a specific class occurs or does not occur according to the model's predictions [37]. One of the most common is *Precision*, also known as the *Positive Predictive Value (PPV)*, which expresses the probability that an instance actually belongs to a given class, given that the model has predicted it as such. Formally, Precision is computed as:

$$\text{Precision} = \text{PPV} = \frac{TP}{TP + FP}.$$

2.2.4.4 Receiver Operating Characteristic

Since conditional measures alone do not always provide a comprehensive understanding of model performance, it is often preferable to employ a visualization method that summarizes their relationship. The **Receiver Operating Characteristic (ROC)** curve is a graphical representation used to evaluate the performance of a binary classification model. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR) across various classification thresholds. Each point on the ROC curve corresponds to a different decision threshold, allowing visualization of the trade-off between Sensitivity (Recall) and Specificity. A model with perfect discriminative ability reaches the top-left corner of the plot, where TPR equals 1 and FPR equals 0. The **Area Under the Curve (AUC)** provides a single scalar value summarizing the ROC curve; it quantifies the model's overall capability to distinguish between positive and negative instances. An AUC of 1 represents perfect classification, while an AUC of 0.5 indicates random performance, equivalent to random guessing.

2.2.4.5 Accuracy

One of the most widely used metrics for evaluating AI-based systems is **Accuracy**, which measures the proportion of correct predictions out of all predictions made. It is defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$

However, in anomaly detection, Accuracy is generally not considered a reliable metric due to dataset imbalance. In most anomaly detection datasets, the number of normal records far exceeds the number of anomalies, leading to misleading results if Accuracy is used alone. Therefore, it is often disregarded in favor of metrics that better reflect model performance under imbalanced conditions.

2.2.4.6 F1 Score

As noted above, Precision and Recall typically exhibit an inverse relationship: increasing Recall by identifying more true positives often results in a decrease in Precision, as more false positives are introduced. To balance these two aspects, the **F1 Score** was introduced. This metric combines Precision and Recall into a single measure that captures the trade-off between them. It is particularly useful when dealing with imbalanced datasets, as it represents the harmonic mean of Precision and Recall and mitigates the limitations of Accuracy in such contexts. The F1 Score is calculated as:

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

From this formula, it is evident that the F1 Score penalizes models that perform well on one metric but poorly on the other. A high F1 Score requires both Precision and Recall to be consistently high. The metric ranges between 0 and 1, with higher values indicating better model performance.

2.2.4.7 Matthews Correlation Coefficient

As discussed above, it is important to account for factors such as dataset imbalance and the trade-off between Precision and Recall. A main limitation of the F1 Score is that it considers only these two metrics, without incorporating the full information available in the confusion matrix. The **Matthews Correlation Coefficient (MCC)** addresses this limitation by taking into account all four values from the confusion matrix, while also being robust to class imbalance. The MCC is defined as:

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

In this equation, the numerator rewards correct classifications while penalizing misclassifications, and the denominator normalizes the result within a range from -1 to +1. A value of +1 indicates perfect predictions, 0 corresponds to random performance, and -1 signifies complete disagreement between predictions and actual labels, representing the worst possible classification outcome.

2.3 Synthetic Data Generation

Data collection from the field can sometimes be challenging. Moreover, the collected data may not be sufficient to train Artificial Intelligence methods, which often require large amounts of data for effective training. To address this issue, **Synthetic Data Generation (SDG)** has emerged as a potential solution. SDG can be defined as the task of generating artificial data and labels with the goal of closely emulating authentic samples.

As outlined by Bauer et al. [4], as Machine Learning and Deep Learning models have evolved in complexity, the limited size of training datasets has become a significant limitation. Consequently, considerable attention has been devoted to SDG methods. In particular, a wide variety of technologies have been explored to produce valuable synthetic data. These methods include both statistical approaches based on traditional Machine Learning techniques and, more recently, Deep Learning and Transformer-based approaches.

In the literature, there exist entire frameworks dedicated to synthetic data generation. One notable example is the **Synthetic Data Vault (SDV)** [43], a system capable of modeling relational databases and sampling synthetic data from them.

In general, comparisons among open-source synthetic data generation systems are still an active area of research. As highlighted by Del Gobbo [14], there is no single superior open-source framework; overall performance tends to be comparable across different systems, and the choice depends largely on the specific tasks and models involved.

2.3.1 Synthetic Data Quality Evaluation

When dealing with synthetic data, it is important to assess the quality of this type of data. In other words, a set of metrics is necessary to provide a quantitative measure of data quality. Specifically, it is essential to understand how to determine this quality, what criteria define data as ”good”, and which methods should be employed to obtain these metrics. However, as reported by Stenger et al. [55], currently, there is no consensus within the research community on how to answer these questions satisfactorily. The evaluation method largely depends on the field of application and the specific tasks for which the synthetic data will be used.

Therefore, in the following sections, we present some Synthetic Data Evaluation Methods that are essential to understand in order to contextualize the work presented here.

2.3.1.1 Visual Representation

Visual representation of real and synthetic data is still considered a valuable method to assess whether a Synthetic Data Generation (SDG) method has performed well. In particular, this representation is not focused on individual samples but rather on the entire dataset. It is possible to visualize both the distributions of real and synthetic data, as well as their representations in a low-dimensional space.

The main limitation of this type of evaluation is that it relies entirely on human judgment [55]. Furthermore, even if the visual representations of real and synthetic data appear very similar, this does not necessarily imply high data quality. In fact, in cases where the SDG method has overfitted, the representations may be identical, but this does not indicate good synthetic data, it simply reflects a replication of the existing real dataset.

Distribution Visualization In the case of distribution visualization, we simply plot the distributions of each feature for both synthetic and real data. This allows us to visually assess whether these distributions are similar. Naturally, the more similar the distributions of real and synthetic data are, the higher the quality of the synthetic data.

By examining these distribution plots, it is possible to determine if the shape of each feature's distribution in the synthetic data matches that of the real data.

Latent Space Visualization Another approach is to represent data in a low-dimensional space; this corresponds to the visual representation of what is called the **latent space**. A low-dimensional space, typically with 2 or 3 dimensions, enables the creation of plots that visually present the data. Ideally, a high overlap between real and synthetic data points is desired. The greater the similarity between the synthetic and real data representations, the higher the quality of the generated data.

To represent data in a low-dimensional space, many algorithms can be employed. The most prominent examples include principal component analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE) [67]. It is also possible to use other algorithms, such as the Uniform Manifold Approximation and Projection (UMAP) visualization algorithm [28].

2.3.1.2 Distance Metrics

Visual representation alone is not sufficient to properly evaluate synthetic data. It is also important to have metrics capable of quantifying quality. In particular, there exist metrics used to assess how similar synthetic data is to real data. In this context, we enter the field of distance metrics.

Maximum Mean Discrepancy One of the most widely used metrics in Synthetic Data Generation (SDG) is the **Maximum Mean Discrepancy (MMD)** [18], a kernel-based statistical test that quantifies the similarity between two probability distributions. MMD equals zero if and only if the two distributions are identical, given that the employed kernel is *characteristic*.

Formally, let P and Q be two probability distributions defined on a common space $\mathcal{X}$, and let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a positive definite kernel associated with a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$. The kernel mean embedding of a distribution P in $\mathcal{H}$ is defined as:

$$\mu_P = \mathbb{E}_{x \sim P}[\, \phi(x) \,],$$

where $\phi : \mathcal{X} \to \mathcal{H}$ is the feature map induced by the kernel $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$.

The *Maximum Mean Discrepancy* between P and Q is then defined as the distance between their mean embeddings in the RKHS:

$$\mathrm{MMD}(P, Q; \mathcal{H}) = \|\mu_P - \mu_Q\|_{\mathcal{H}}.$$

Intuitively, MMD measures how far apart the mean embeddings of two distributions are in the RKHS. A smaller value of MMD indicates higher similarity between distributions, while a larger value signals greater dissimilarity.

Earth Mover's Distance The **Earth Mover's Distance (EMD)**, also known as the first-order *Wasserstein distance*, provides a geometric approach to compare two discrete distributions (or sets of weighted features) by modeling the problem as an instance of the optimal transportation problem.

Given two *signatures*:

$$P = \{(p_i, w_{p_i})\}_{i=1}^{m}, \quad Q = \{(q_j, w_{q_j})\}_{j=1}^{n},$$

where $p_i, q_j \in \mathcal{X}$ are the support points and $w_{p_i}, w_{q_j} \geq 0$ their associated weights (summing to the total mass of each distribution), we define the ground distance matrix:

$$d_{i,j} = d(p_i, q_j), \quad 1 \le i \le m,\ 1 \le j \le n.$$

The goal is to find a flow matrix $F = [f_{i,j}]$ that minimizes the total transportation cost:

$$\min_{f_{i,j}} \sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j}\, d_{i,j}$$

subject to the following constraints:

$$\begin{cases} f_{i,j} \ge 0, & \forall\, i, j, \\ \sum_{j=1}^{n} f_{i,j} \le w_{p_i}, & \forall\, i, \\ \sum_{i=1}^{m} f_{i,j} \le w_{q_j}, & \forall\, j, \\ \sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j} = \min\left(\sum_{i=1}^{m} w_{p_i},\ \sum_{j=1}^{n} w_{q_j}\right). \end{cases}$$

Once the optimal flow $F^* = [f^*_{i,j}]$ is obtained, the *Earth Mover's Distance* is defined as the normalized minimal cost:

$$\mathrm{EMD}(P, Q) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} f^*_{i,j}\, d_{i,j}}{\sum_{i=1}^{m} \sum_{j=1}^{n} f^*_{i,j}}.$$

Intuitively, $f^*_{i,j}$ represents the optimal amount of "mass" transported from p_i to q_j, and $d_{i,j}$ denotes the ground distance between them. The EMD thus corresponds to the minimum average work required to transform one distribution into the other.

Dynamic Time Warping In the case of temporal sequences, i.e., when dealing with time series, it can be useful to have a metric capable of evaluating the temporal similarity between two sequences. For this purpose, **Dynamic Time Warping (DTW)** [54] is a good choice. This metric allows computing the temporal similarity between two sequences even if they have different lengths or vary in speed, by non-linearly warping one sequence to match the other.

Let two sequences:

$$X = (x_1, x_2, \dots, x_N), \quad Y = (y_1, y_2, \dots, y_M)$$

be given (of possibly different lengths N and M). We define a cost (or distance) function:

$$d(x_i, y_j)$$

that measures the local distance between elements x_i and y_j. Construct the cost matrix $D \in \mathbb{R}^{N \times M}$ whose entries are

$$D_{i,j} = d(x_i, y_j), \quad 1 \le i \le N,\ 1 \le j \le M.$$

A warping path is a sequence:

$$W = \big((i_1, j_1), (i_2, j_2), \dots, (i_L, j_L)\big)$$

of matrix indices satisfying the conditions:

$$\begin{aligned}
&i_1 = 1,\ j_1 = 1,\\
&i_L = N,\ j_L = M,\\
&i_{k+1} - i_k \in \{0,1\},\ j_{k+1} - j_k \in \{0,1\},\ (i_{k+1} - i_k) + (j_{k+1} - j_k) \ge 1,
\end{aligned}$$

for $k = 1, \dots, L-1$. The cumulative cost of a path W is:

$$\mathrm{Cost}(W) = \sum_{k=1}^{L} D_{i_k, j_k}.$$

Then the DTW distance between X and Y is defined as the minimal cumulative cost over all valid warping paths:

$$\mathrm{DTW}(X, Y) = \min_{W} \ \mathrm{Cost}(W).$$

In practice one uses dynamic programming with the recurrence:

$$\begin{aligned} \gamma(i, j) &= D_{i,j} + \min\{\gamma(i-1, j),\, \gamma(i, j-1),\, \gamma(i-1, j-1)\}, \\ \gamma(1, 1) &= D_{1,1}, \\ \mathrm{DTW}(X, Y) &= \gamma(N, M). \end{aligned}$$

Here $\gamma(i, j)$ denotes the minimal cumulative cost to match the prefixes $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$.

Thus, due to its structure, this algorithm provides a similarity measure that is invariant to time shifts. In this way, it is able to take into account temporal similarities, or patterns, that occur at different speeds.

2.3.1.3 Discriminator Two-Sample Test

The Classifier Two-Sample Test, also known as the **Discriminator Two-Sample Test (D2ST)**, is not a direct metric to evaluate the similarity between two datasets. Instead, it has been introduced as a method to assess whether two sets of data points are sampled from the same distribution [31].

The method relies on the use of a binary classifier trained to discriminate whether data belongs to one class or the other. In the context of synthetic data evaluation, the binary classifier is trained to distinguish between real and synthetic data. Afterwards, the classifier is tested on a test dataset composed of both real and synthetic samples (Figure 2.5).

Ideally, if the synthetic data is extremely similar to the original data, the classifier should not be able to discriminate between the two datasets; thus, it should perform poorly in terms of classification accuracy. Conversely, if the classifier performs well, it likely indicates that the differences between

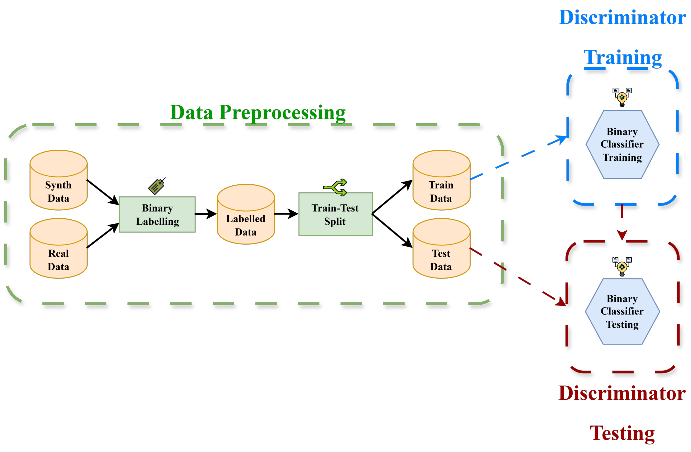


Figure 2.5: *Discriminator Two-Sample Test (D2ST)* flow chart. Synthetic and real data undergo a preliminary preprocessing phase involving a binary labeling procedure. The resulting combined dataset is subsequently partitioned into two subsets: a training set and a test set. A binary classifier is then trained on the training subset and evaluated on the test subset, with the objective of distinguishing between real and synthetic samples.

real and synthetic data are significant. Therefore, the lower the accuracy of the binary classifier, the higher the similarity between the two datasets, which implies better quality of the synthetic data.

Additionally, this method has the advantage, at the sample level, of directly identifying which samples are fooling the classifier [55].

2.3.2 Synthetic Data Generation Methods

As introduced before, there exist a great number of synthetic data generation methods. In the following, we present some of them, highlighting the core idea behind each method.

2.3.2.1 Probabilistic and Statistical Models

The first family of synthetic data generation (SDG) methods we introduce is the group of so-called **Probabilistic and Statistical Models**. These models are based on the idea of emulating the probability density function (PDF) of the data in order to sample from it and produce new synthetic data similar to the original.

GMMs Gaussian Mixture Models (GMMs), introduced earlier, are able to model the PDF of real-world data using a certain number of Gaussian components. Once a GMM is fit on the data, it is possible to sample from it to use as a generative probabilistic model [55].

In practice, synthetic data can be produced from a trained GMM by randomly selecting a Gaussian component according to the probability of that specific component, then sampling from the normal distribution of the selected component (Figure 2.6).

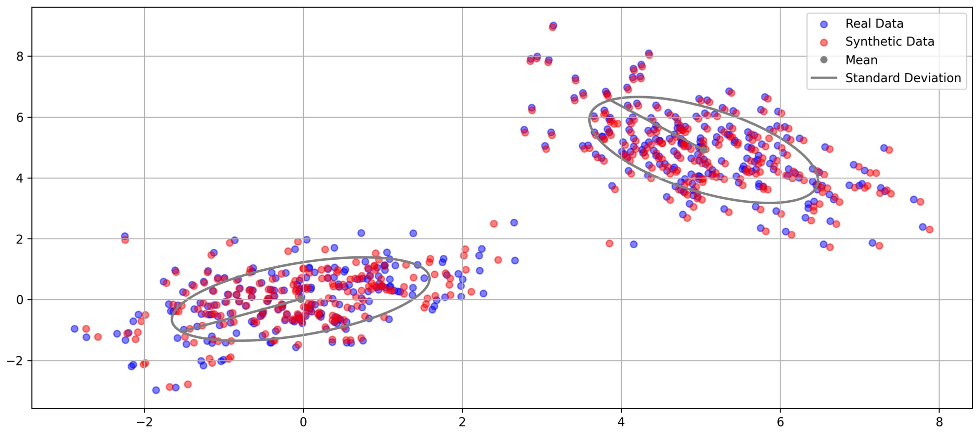


Figure 2.6: Scatter plot of real and synthetic data sampled from a two-component Gaussian Mixture Model. The ellipses represent the 95% confidence regions of the Gaussian components, while the lines indicate the direction and magnitude of the principal standard deviation.

Probabilistic Autoregressive (PAR) In the realm of neural networks, it is possible to design models capable of learning the probability distribution underlying a given dataset. In particular, a **Probabilistic Autoregressive (PAR)** neural network learns the conditional probability distribution of future observations in a time series given past values [41]. These models exploit neural architectures to capture nonlinear relationships while relying on the autoregressive principle to model temporal dependencies.

The goal of PAR models is not to output a single deterministic forecast, but rather to estimate the parameters, typically mean and variance, of the predictive distribution for the next time step. Formally, given a sequence $\{y_t\}$, a PAR model learns:

$$p(y_{t+1} \mid y_t, y_{t-1}, \dots, y_{t-u}),$$

where the parameters of the distribution (e.g., μ_{t+1} and σ_{t+1}) are predicted via a neural network. A general formulation is:

$$(\mu_{t+1}, \sigma_{t+1}) = f_\theta(y_t, \dots, y_{t-u}),$$

where f_θ denotes a nonlinear function implemented by the network and parameterized by θ.

A widely adopted backbone for PAR models is the class of **Autoregressive Neural Networks (ARNN)**, designed for modelling nonlinear time series [69]. An ARNN(u, v) receives u lagged observations as inputs and processes them through a single hidden layer consisting of $v = \lceil (u+1)/2 \rceil$ neurons. After training with gradient-based backpropagation, the forecast is obtained from the nonlinear transformation:

$$g(y) = \alpha_0^* + \sum_{j=1}^{v} \alpha_j^* \, \phi\Big(\beta_j^* + \theta_j^{*\prime} y\Big) \,,$$

where y is the u-lagged input vector, $\phi(\cdot)$ is a bounded activation function, and $\alpha_j^*, \beta_j^*, \theta_j^*$ are the network weights.

Recent developments have led to the definition of **Probabilistic Autoregressive Neural Networks (PARNN)**[41], which extend ARNN models by incorporating residual information from an ARIMA component. A PARNN(m, k, l) predicts future values as a nonlinear function of m lagged observations and l lagged ARIMA residuals (feedback errors). This yields:

$$y_t = f(y_{t-1}, \dots, y_{t-m}, e_{t-1}, \dots, e_{t-l}),$$

where f is a single-hidden-layer ARNN with $k = \lceil (m + l + 1)/2 \rceil$ neurons. The explicit formulation is:

$$y_t = \alpha_0 + \sum_{j=1}^{k} \alpha_j \, G\left(\alpha_{0,j} + \sum_{i=1}^{m} \alpha_{i,j} y_{t-i} + \sum_{i=m+1}^{m+l} \alpha_{i,j} e_{t+m-i} \right) + \epsilon_t,$$

where $G(\cdot)$ is a sigmoidal activation function. Due to this structure, PARNN models are well-suited to time series exhibiting non-stationary, nonlinear, and non-Gaussian patterns.

2.3.2.2 Similarity-Based Models

Another group of Synthetic Data Generation methods is what we call **Similarity-Based Techniques**. These models or algorithms produce new samples in a deterministic way, without learning probability distributions of the data. Instead, they generate ”similar” datapoints based on mathematical assumptions.

Synthetic Minority Over-sampling Technique (SMOTE) One of the most famous methods to produce synthetic data is the **Synthetic Minority Over-sampling Technique (SMOTE)** [5], originally introduced to address the common problem of class imbalance in real datasets. This method oversamples the minority class to rebalance the dataset.

The algorithm operates not in the original ”data space,” but in the ”feature space.” Given an imbalanced dataset, SMOTE oversamples the minority class by taking each datapoint in this class and computing its k nearest neighbors. Then, one neighbor is randomly selected from these k neighbors. The difference between the two points is computed, multiplied by a random number between 0 and 1, and added to the original datapoint. This process generates a new synthetic sample lying between the two selected points (Figure 2.7). The procedure is repeated until the desired number of synthetic samples is generated.

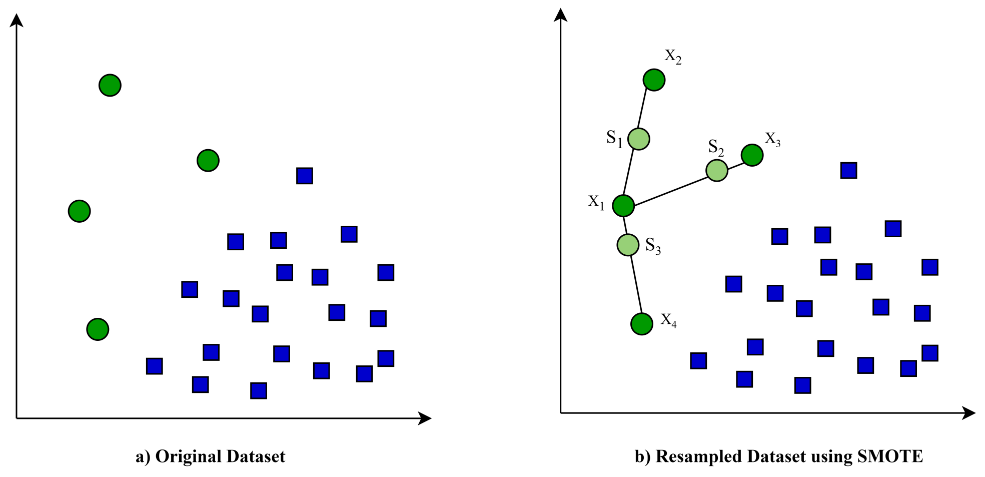
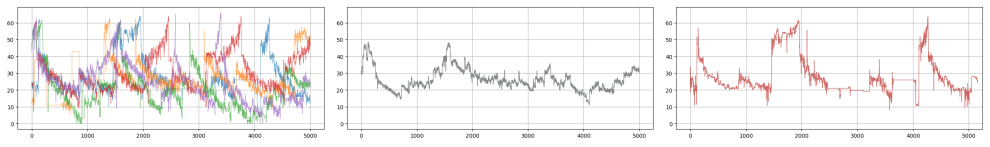


Figure 2.7: SMOTE algorithm example: the left plot shows the original dataset, and the right plot the resampled dataset with synthetic samples. The synthetic point S_1 is generated by interpolating between a minority sample x_1 and one of its nearest neighbors x_2 as $S_1 = x_1 + \delta(x_2 - x_1)$, where $\delta \in [0, 1]$ is a random scalar.

Dynamic Time Warping Barycenter Averaging (DTWBA) As introduced earlier, Dynamic Time Warping (DTW) is a metric highly useful for measuring similarity between two time series in terms of temporal patterns. One key challenge when working with metrics like DTW is defining an averaging technique consistent with the metric.



(a) RSSI Time Series. **(b)** Arithmetic Mean. **(c)** DTWBA Generated Time Series.

Figure 2.8: Comparison between the classical arithmetic mean of the time series and the representative sequence obtained through DTW Barycenter Averaging (DTWBA).

Petitjean et al. [45] proposed the **Dynamic Time Warping Barycenter Averaging (DTWBA)** method to address this problem. Instead of iterative pairwise averaging, their approach builds a metric invariant to the order of data. The algorithm iteratively refines an initial average sequence to minimize its squared DTW distance to the sequences being averaged. This iterative process converges because at each step the inertia (sum of squared distances) can only decrease, as the new average sequence is closer to the elements it averages. In Figure 2.8 is shown the comparison between a classical arithmetic mean of time series and a sequence obtained using DTWBA, from the image it is posibble to see that DTWBA is capable to map the temporal dependencies inside the time series.

DTWBA computes a barycentric average of a set of time series by iteratively refining an initial prototype sequence. Let $\mathcal{S} = \{\vec{S}_1, \dots, \vec{S}_N\}$ be a collection of N sequences of length T, and let $\vec{A}^{(i)} = (A_1^{(i)}, \dots, A_T^{(i)})$ denote the average at iteration i. The method proceeds in two phases at each iteration:

1. **Alignment step:** DTW is computed between each $\vec{S}_n$ and the current average $\vec{A}^{(i)}$. For each position t of the average sequence, this yields a set of associated coordinates

$$\text{assoc}(A_t^{(i)}) = \{x_j \mid x_j \text{ is aligned with } A_t^{(i)} \text{ under DTW}\}.$$

2. **Update step:** Each coordinate is updated via the barycenter operator:

$$A_t^{(i+1)} = \text{barycenter}\Big(\text{assoc}(A_t^{(i)})\Big),$$

where, for a set of m aligned coordinates $X_1, \dots, X_m$,

$$\text{barycenter}(X_1, \dots, X_m) = \frac{1}{m}\sum_{j=1}^{m} X_j.$$

This iterative refinement minimizes the global inertia, i.e. the sum of squared DTW distances between the current average and all sequences in the set. Since each update reduces inertia, the process converges.

Furthermore, DTWBA is not only an averaging method but can also be used to generate synthetic time series [12]. Given a set of time series, the global average computed by DTWBA can serve as a synthetic time series. By introducing weights and varying them during the averaging process, it is possible to generate an infinite number of synthetic time series from the original set.

2.3.2.3 Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GAN) [15] are a class of deep learning models designed to generate synthetic data by learning the underlying data distribution. The framework consists of two neural networks trained simultaneously: a *generator* G and a *discriminator* D, both commonly implemented as multilayer perceptrons. The generator maps a noise vector z, sampled from a prior distribution $p_z(z)$, to the data space, producing samples $G(z; \theta_g)$ with parameters θ_g. The discriminator $D(x; \theta_d)$, with parameters θ_d, outputs a scalar representing the probability that input x originates from the true data distribution rather than from the generator.

Training proceeds as a two-player minimax game defined by the value function:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))].$$

The discriminator is optimized to correctly classify real and generated samples, while the generator aims to produce samples that maximize the discriminator's error, effectively "fooling" it (Figure 2.9).

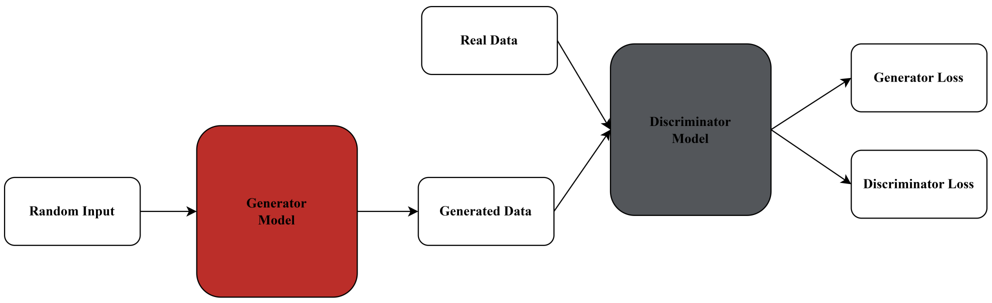


Figure 2.9: General architecture of a Generative Adversarial Network (GAN) model.

In practice, the optimization alternates between updating the discriminator for several steps and updating the generator once, maintaining the discriminator near optimality as the generator gradually improves. This iterative training avoids overfitting and computational inefficiency associated with fully optimizing the discriminator at each iteration.

Because the original generator loss $\log(1 - D(G(z)))$ can saturate early in training, an alternative objective maximizing $\log D(G(z))$ is often used to provide stronger gradients and improve convergence. Theoretically, given sufficient model capacity, this adversarial training allows the generator to recover the true data distribution in the non-parametric limit.

Conditional Tabular GAN (CTGAN) One common challenge in GAN-based models is to simultaneously model continuous and discrete variables and to handle imbalance in categorical features. To address these issues, **Conditional Tabular GAN (CTGAN)** was introduced [65]. This model differs from other GAN-based methods by introducing mode-specific normalization during training, architectural changes, and the ability to address data imbalance via a conditional generator.

In real data, discrete features are often imbalanced; training a GAN on such data risks poor representation of minority categories. Consequently, the generator may fail to learn to generate these categories effectively. CTGAN solves this by introducing conditional generation, allowing the generator to explore all categories, including minority ones. The generation is conditional because a condition vector guides this exploration.

However, this approach risks generating data that do not represent the real distribution due to resampling. To mitigate this, the authors introduced ”*training-by-sampling*,” which samples conditions in a balanced way based on the log-frequencies of the data. This ensures the generator sees data from all categories during training. By combining training-by-sampling and conditional generation, CTGAN guarantees synthetic data that are both representative of real distributions and of good quality for minority categories.

DoppleGANger Standard GAN-based models are not directly designed to handle temporal data and cannot effectively learn temporal dependencies in time series. To address this, **DoppleGANger** was introduced [29]. This model leverages GAN technology to handle both continuous and discrete features and is designed to learn temporal patterns in data.

Typically, GAN generators use Multi-layer Perceptrons (MLPs), which are not well suited to capture long-term correlations in temporal data. DoppleGANger replaces the MLP generator with Recurrent Neural Networks (RNNs), which are designed for temporal data. The discriminator, however, remains an MLP because the use of the Wasserstein distance as a loss function requires second derivative computations, which current libraries do not support with RNNs.

A major challenge of DoppleGANger is that when the number of features is large, data fidelity decreases significantly. To address this, DoppleGANger employs an auxiliary discriminator that focuses only on attributes rather than all features. This helps the generator learn to produce high-quality attributes and features effectively.

2.3.2.4 Autoencoder-Based Models

As introduced before, an autoencoder is a neural network architecture composed of two parts: the encoder and the decoder. Since the autoencoder is used to fit the data distribution, it is possible to use an autoencoder-based architecture to learn distributions and reproduce them. Therefore, autoencoders can be used to produce synthetic datasets. It is important that the autoencoder model learns the distributions and not just reproduces data; only in this way the model will be able to generate new synthetic data similar to the original but not exact copies or real datapoints.

Tabular Variational Autoencoder (TVAE) Within the family of autoencoders, a particular type widely used in the field of Synthetic Data Generation (SDG) is the state-of-the-art **Variational Autoencoder (VAE)** [26]. From a mathematical point of view, a VAE is a probabilistic model that aims to model the distribution of input data x. Specifically, its task is to learn a parametrized distribution $p_\theta(x)$, where θ are the parameters, maximizing the likelihood of observing data following this distribution. This is the main difference with respect to a classical autoencoder, which models data deterministically rather than probabilistically.

We assume that the data x depends also on latent (hidden) variables z, which represent the latent distribution of the data itself. To obtain the distribution of data, we integrate over all latent variables:

$$p_\theta(x) = \int_z p_\theta(x, z)\, dz$$

where $p_\theta(x, z)$ is the joint distribution under p_θ of the observable data x and the latent representation z. By the chain rule, this can be rewritten as:

$$p_\theta(x) = \int_z p_\theta(x|z)\, p_\theta(z)\, dz$$

where $p_\theta(z)$ is the prior distribution and $p_\theta(x|z)$ is the conditional probability of x given z.

There are two main problems related to this formulation:

- *Intractability*: the posterior distribution makes the integral intractable and difficult to compute, preventing the use of the Expectation-Maximization algorithm.

- *Large datasets*: in case of large amounts of data, batch optimization is computationally costly and time-consuming [26].

To overcome these issues, VAE relies on a so-called ”*recognition model*” $q_\phi(z|x)$, which approximates the intractable true posterior $p_\theta(z|x)$ and is parametrized by ϕ. Since z represents the latent code of x, the recognition model $q_\phi(z|x)$ can be seen as a probabilistic encoder: given a datapoint x, it produces a distribution over possible values of the code z from which x could have been generated. Similarly, $p_\theta(x|z)$ can be seen as the decoder, producing a distribution over possible values of x given a code z.

VAEs are usually trained by optimizing the **Evidence Lower Bound (ELBO)** loss function, which consists of two terms. The first term measures the reconstruction ability of the model, balancing how well the decoder reconstructs real data x from the code z. The second term is a regularization term that balances the similarity between the approximate posterior $q_\phi(z|x)$ and the prior distribution.

When dealing with tabular data and variational autoencoders, the state-of-the-art model is the **Tabular Variational Autoencoder (TVAE)**, implemented by Xu et al. [65], where the authors adapted the VAE model by modifying the loss function to better handle tabular data.

2.3.2.5 Score-based Diffusion Models

Another generative family of models is the so-called **Diffusion Models (DF)**. These models generate new data by learning the *diffusion process* of the given dataset. The idea is that a new datum can be generated by performing a step inside the space of all possible data. These models are Markov chains that iteratively add Gaussian noise to data in a forward process over a certain number of steps. They are also able to learn the reverse process, mapping the noise back to the original data distribution [4].

More deeply, diffusion models are Markov chains that iteratively add Gaussian noise to data in a forward process over T steps and also learn the reverse process that iteratively maps the noise input back to the data distribution. Note that, given a datum x_0 and a number of steps T, the forward diffusion process gradually adds noise to the data according to the Markov chain:

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}),$$

where each transition adds Gaussian noise:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\Big(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\Big),$$

with variance schedule $\{\beta_t\}_{t=1}^{T}$, which can be learnable or constant.

The reverse (sampling) process is modeled as another Markov chain with learned transitions:

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t),$$

where each reverse transition is parameterized as a conditional Gaussian:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\Big(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)\Big).$$

The model is trained by variational inference to approximate the true reverse diffusion process.

ForestDiffusion In the majority of state-of-the-art diffusion-based models, the main architectures used are based on neural networks. However, Jolicoeur-Martineau et al. [25] introduced a model called **Forest Diffusion**, which is based not on neural networks but on a Gradient-Boosted Trees (GBTs) model called **XGBoost** [7].

Recall that decision trees are predictive models that partition input data into distinct subsets via decision splits, selecting splits to maximize predictive performance. Gradient-Boosted Trees use a sequence of decision trees, where each new tree aims to correct the errors made by the previous trees during prediction. The process starts with one tree, then sequentially adds others that emphasize samples previously predicted incorrectly. This continues until a predefined number of trees is reached.

As mentioned, traditional diffusion and flow models rely exclusively on deep neural networks to estimate the vector field or score function. In contrast, Forest Diffusion employs GBTs, specifically XGBoost, as the underlying model for this estimation. This approach requires a different training methodology because GBTs do not use Stochastic Gradient Descent (SGD) and instead require training on a fixed dataset.

To approximate the expectations over data and noise pairs inherent in diffusion model losses, the original training dataset is duplicated multiple times. Each duplicate is paired with independently sampled Gaussian noise, creating an expanded dataset of noisy samples and corresponding target values (the entire process is summurized in Figure 2.10). This precomputed dataset enables XGBoost to be trained effectively without stochastic mini-batch sampling.

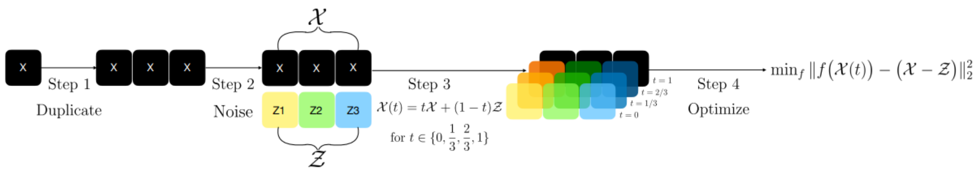

$\mathcal{X}$
X
X X X
X X X
Step 1
Duplicate
Step 2
Noise
Z1 Z2 Z3
$\mathcal{Z}$
Step 3
$\mathcal{X}(t) = t\mathcal{X} + (1-t)\mathcal{Z}$
for $t \in \{0, \frac{1}{3}, \frac{2}{3}, 1\}$
$t=1$
$t=2/3$
$t=1/3$
$t=0$
Step 4
Optimize
$\min_f \|f(\mathcal{X}(t)) - (\mathcal{X} - \mathcal{Z})\|_2^2$

Figure 2.10: Workflow of Forest Diffusion from original paper: *"Generating and Imputing Tabular Data via Diffusion and Flow-based Gradient-Boosted Trees"*[25]. The process is based on four steps: In the first the dataset is duplicated, in the second for each duplicated dataset $\mathcal{X}_\rangle$ a noise $\mathcal{Z}_\rangle$ is added. In the third step the linear interpolation between the duplicated dataset and their corresponding noise for different time t is computed. In the last step an optimization process is performed on GBTs models (one for each noise level).

A critical hyperparameter in this process is the number of noise samples per data point, which directly affects the quality of the expectation approximation. The authors suggest setting this value to 100, balancing performance and memory constraints [25].

2.3.2.6 Transformer-based models

Transformers are a class of sequence-to-sequence transduction models that have revolutionized the field of natural language processing (NLP) by introducing an architecture fundamentally based on the *attention mechanism*. Unlike traditional recurrent or convolutional neural networks, transformers can directly model dependencies between any two positions in the input sequence, regardless of their distance. This capability allows them to overcome limitations related to long-range dependency modeling and parallelization inefficiencies.

Originally proposed by Vaswani et al. [61] for NLP tasks, transformers have demonstrated outstanding performance in a variety of applications such as text generation, summarization, and machine translation. The core innovation of the transformer architecture is the *self-attention* mechanism, which enables the model to compute, for each element of the input sequence, a weighted representation of all other elements. This dynamic weighting allows the model to focus selectively on relevant parts of the input when producing outputs.

The transformer model consists of two main components: an encoder and a decoder, each composed of a stack of L identical layers. Each encoder layer contains two primary sub-modules: a multi-head self-attention mechanism and a position-wise feed-forward network (FFN). To facilitate training of deep networks, residual connections wrap each sub-module, followed by layer normalization.

The self-attention mechanism operates by projecting the input embeddings into three distinct matrices: queries $\mathbf{Q}$, keys $\mathbf{K}$, and values $\mathbf{V}$. Given these, attention weights are computed via a scaled dot-product:

$$\mathrm{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V},$$

where d_k is the dimension of the keys, and the scaling factor $\sqrt{d_k}$ helps stabilize gradients during training. This operation produces a weighted sum of the values, where weights reflect the relevance between queries and keys.

To enrich the model's ability to attend to information from different representation subspaces, transformers employ *multi-head attention*, which runs multiple attention mechanisms in parallel. Formally, for H heads, the inputs are linearly projected into different query, key, and value spaces, attention is computed independently for each head, and the results are concatenated and projected back:

$$\mathrm{MultiHeadAttn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_H) W_O,$$

$$\text{where} \quad \mathrm{head}_i = \mathrm{Attention}(\mathbf{Q}W_Q^i, \mathbf{K}W_K^i, \mathbf{V}W_V^i).$$

The decoder layers extend the encoder structure by including *masked self-attention* to prevent positions from attending to future tokens during autoregressive generation, and *cross-attention* modules that attend to encoder outputs, integrating contextual information from the input sequence.

Since transformers lack any inherent notion of token order (no recurrence or convolution), *positional encodings* are added to the input embeddings to provide explicit information about the position of tokens in the sequence. The entire architecture is shown in figure 2.11.

Each layer also contains a fully connected feed-forward network applied independently at each position:

$$\mathrm{FFN}(H') = \mathrm{ReLU}(H'W_1 + b_1)W_2 + b_2,$$

where W_1, W_2 and biases b_1, b_2 are learned parameters, and the intermediate dimension is typically larger than the model dimension. Residual connections and layer normalization are applied around both the attention and feed-forward sub-modules to stabilize and accelerate training:

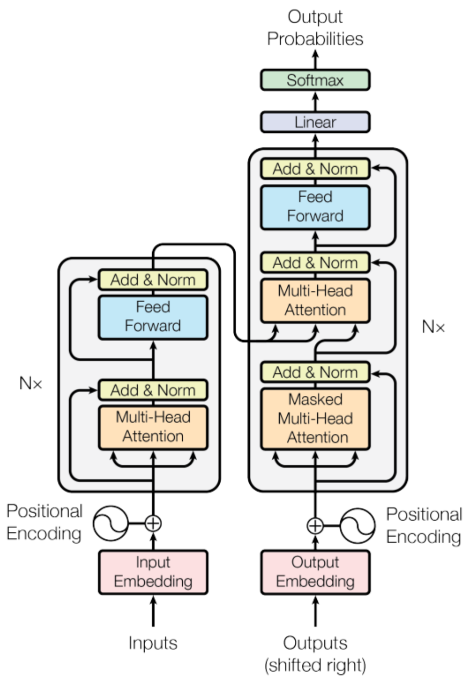


Figure 2.11: Transformer architecture from original paper: ”Attention is all you need”[61].

$$H' = \mathrm{LayerNorm}(\mathrm{SelfAttention}(X) + X),$$

$$H = \mathrm{LayerNorm}(\mathrm{FFN}(H') + H').$$

Beyond NLP, transformer architectures have been successfully adapted to other domains involving sequential or structured data. For instance, Parmar et al. [42] proposed the *Image Transformer*, an encoder-decoder model designed for high-resolution image generation. This approach formulates image synthesis as a sequence modeling problem, where the RGB values of pixels are predicted one by one based on the context provided by neighborhood pixels.

Nowadays, transformer-based methods are used to generate more or less any type of data, such as text, images, video [56], and time series [2] [9].

Chronos Time series represent a particular type of data, extremely challenging for some SDG models because it is difficult for these models to capture the temporal patterns in data. Now, if we think about natural language, we can consider it just like a sequence of words (or tokens) in time; thus, we can model language like a univariate time series, where the attribute is a token that changes over time, building sentences. If the attention mechanism is able to map long-term dependencies in sentences, then it means that maybe it is also able to map long-term dependencies in temporal series. This is the idea behind a framework called **Chronos** [2].

Chronos is a framework which presents some transformer-based models of the T5 family [50], pretrained for time series forecasting. The idea is essentially that it is possible to forecast the next time series values, like it is possible to forecast the next token in sentences.

As reported by the authors:

> Chronos tokenizes time series into discrete bins through simple scaling and quantization of real values. In this way, we can train off-the-shelf language models on this "language of time series," with no changes to the model architecture.

Thus, the model architecture is more or less untouched.

Note that, since the framework uses transformer-based models, it is clear that input data has to be tokenized. For time series forecasting, it is necessary to map the observations to a finite set of tokens. To this end, the framework scales and then quantizes observations into a fixed number of bins. In particular, the scaling process relies on *mean scaling*, which normalizes individual entries of the time series by the mean of the absolute values in the historical context. But once scaled, the series is composed of real values that cannot be given as input to language models. So, a quantization mechanism is required.

The quantization mechanism is as follows: a given number of bin centers on the real line is selected; then, for each bin also the edges are chosen. This binning technique is uniform, so the width of bins is equal for each bin. Also, two other tokens are used: `PAD` to replace missing values and to pad time series to a fixed length, and `EOS` to signal the end of the tokenized and padded time series.

The main limitation of Chronos relies on the binning technique. In fact, if the prediction range is limited to an interval given by the first and the last bin centers, then it is obvious that theoretically the model is not able to deal with time series with a strong trend.

Time Series Foundation Model (TimesFM) In order to build a zero-shot model able to forecast time series, Das et al. [9] introduced a foundational model based on a decoder-only architecture, called **TimesFM**. This model relies on a *patching* procedure during training. In fact, the data is partitioned into patches, where the patch is considered as the analogue of the token in natural language processing. Thus, the task of the decoder-only model is to learn to forecast the following patch given the previous ones. This model is interesting because it is not directly trained only to predict the following patch, but it is also trained to predict a bunch of patches of different lengths. In fact, during training, the length of patches is changed in order to train the model to predict time series of different lengths. Note also that a *masking* procedure is used to train the model to learn all possible context lengths starting from 1 to a maximum length. The presented model is called **Time-series Foundation Model (TimesFM)**.

Since the model is trained to predict the next patch of data, a point forecasting loss is used during training; in particular, the Mean Squared Error (MSE) is used.

The preprocessing of input data is similar to the one seen for Chronos: the input time series is processed into tokens by the input layers dividing the data into contiguous non-overlapping patches. Then each patch is processed by a residual block into a vector of a size given as parameter. The input is masked with a binary mask that specifies if the corresponding input should be ignored or not. Then, the vector is added to positional encodings and fed into a certain number of stacked transformer layers with multi-head attention. On top of these transformer layers, a feed-forward network composes the fully connected layer that outputs the probabilities over the entire vocabulary. The output tokens are then mapped through a residual block which is the forecast for the time window following the last input patch seen by the model.

2.4 Constrained Optimization and SMT

In the field of mathematical optimization, *constrained optimization*, or *constraint programming (CP)*, refers to the process of finding feasible solutions from a set of possible solutions of a mathematical problem that is modeled using variables and constraints [51].

Note that CP is based on the idea of finding solutions (in general) and not necessarily on optimization (which means finding the optimal solution). Thus, given a problem, CP allows us to model it and find at least one solution. If the best solution is required, it is also possible to model an *objective function* and minimize or maximize it to find the best possible solution.

More formally, we can say that a CP problem is composed of:

- a set of n *variables* $X = \{x_1, \dots, x_n\}$;
- a set of current domains $D = \{D(x_1), \dots, D(x_n)\}$, where $D(x_i)$ represents the finite set of possible values for variable x_i;
- a set C of constraints between variables.

Note also that the problem is formulated by assigning an initial domain D_0 to each variable. Thus, we can build a set of initial domains $D_0 = \{D_0(x_1), \dots, D_0(x_n)\}$. Moreover, we can define more precisely a constraint C on a subset of the variables X. Thus, if we have the subset $X(C) = \{x_1, \dots, x_r\}$, a constraint is a subset $T(C)$ of the Cartesian product $D_0(x_1) \times \cdots \times D_0(x_r)$ that specifies the allowed combinations of values for the variables $x_1, \dots, x_r$. Formally, an element of $T(C)$ is called a *tuple* on $X(C)$; while the cardinality of $X(C)$

($|X(C)|$) is called the *arity* of constraint C. In fact, constraints can be *unary* (involving a single variable), *binary* (involving two variables), or *global* (involving many variables). Global constraints, such as `AllDifferent` (ensuring all variables take distinct values), capture common patterns and allow specialized, efficient propagation algorithms.

To introduce some important concepts, such as *propagation* and *validity*, we need to define some notation:

- (x, a) denotes that the value a is assigned to variable x;
- $\tau[k]$ denotes the k-th value of the tuple τ.

We can now introduce the concept of *validity*. We say that a tuple τ is **valid** if

$$\forall (x, a) \in \tau, \quad a \in D(x).$$

The concept of *consistency* applies to constraints. In fact, we say that C is **consistent** if and only if there exists a tuple $\tau \in T(C)$ which is valid.

It is also possible to introduce the concept of *consistency with respect to assignments* given to variables. First, we need to define the concept of *support*. We say that a tuple $\tau \in T(C)$ is a **support** for (x, a) on C if it involves the assignment (x, a). Thus, a value $a \in D(x)$ is **consistent with** C if either $x \notin X(C)$ or there exists a valid support for (x, a) on C (i.e., a valid tuple τ such that $(x, a) \in \tau$).

Another important concept is *arc consistency*. We say that a constraint is **arc consistent** if and only if

$$\forall x \in X(C), \quad D(x) \neq \emptyset$$

and

$$\forall a \in D(x), \quad a \text{ is consistent with } C.$$

Clearly, in order to solve a formulated problem, it is necessary to eliminate all the values that are not consistent with a constraint C. Thus, a **filtering algorithm** associated with a constraint C aims to eliminate values that are inconsistent with C without removing those that are consistent. Note that, if this algorithm is able to eliminate all inconsistent values, then we say that C is domain consistent and that the algorithm enforces arc consistency of C.

In CP, the solver's goal is to find a solution to the problem, or the optimal one if an objective function is to be maximized or minimized. The basic idea behind the functioning of these solvers is the concept of **constraint propagation**, a process that reduces variable domains by iteratively enforcing local consistency conditions. In fact, propagation is the mechanism that consists of calling the filtering algorithms associated with the constraints involving a variable x each time the domain of this variable is modified. When domains are finite, this process takes a finite amount of time to find the solution.

In some problems, the search space can be extremely large, so CP solvers are designed to use what are called *search strategies*: systematic search algorithms such as backtracking combined with heuristics to decide the order of variable assignments and value selections, further improving performance.

Satisfiability Modulo Theories (SMT) Modeling problems as Constraint Programming (CP) instances is not always the most effective approach; in some cases, it is more relevant to determine whether a mathematical formula is satisfiable, that is, whether there exists at least one solution that satisfies the formula. When every variable in the formula is Boolean, this problem is known as the *Boolean satisfiability problem (SAT)*. In essence, SAT asks whether there exists an assignment of Boolean variables that makes a given propositional logic formula true [36]. It is important to note that Boolean satisfiability is proven to be an *NP-complete* problem.

Since using only Boolean variables is often insufficient, an extension of SAT called **Satisfiability Modulo Theories (SMT)** has been introduced. SMT extends SAT by allowing formulas to include more complex elements such as real numbers, integers, and various data structures, in other words, it incorporates additional theories. From this perspective, SMT can be viewed as a constraint satisfaction problem and thus relates closely to CP.

SMT solvers combine a SAT solver with one or more specialized theory solvers. The SAT solver manages the Boolean structure of the formula, while the theory solvers reason about constraints within their respective theories, such as linear, arithmetic or arrays. The interaction between these components typically follows a *lazy* approach: the SAT solver proposes assignments to Boolean variables that correspond to theory literals, and the theory solvers then check the consistency of these assignments with respect to their theories. When a conflict is detected, a conflict clause is generated and sent back to the SAT solver to prune the search space. This collaborative process continues until a satisfying assignment is found or unsatisfiability is proven.

The entire search procedure relies on an algorithm called **conflict-driven clause learning (CDCL)**, which is used for SAT problems. CDCL operates by iteratively assigning truth values to Boolean variables and propagating these assignments until either a satisfying assignment is found or a conflict (contradiction) arises. Upon detecting a conflict, the algorithm analyzes it to learn a new clause that prevents the same conflict from recurring and then backtracks non-chronologically to an earlier decision level.

In detail, let F be a conjunctive normal form (CNF) formula over variables V. The solver maintains a partial assignment A that maps some variables in V to $\{\text{true}, \text{false}\}$, a decision level $d \in \mathbb{N}$ initially set to zero, and an implication graph $G = (N, E)$, where nodes N represent assigned literals and edges E represent propagation dependencies. At each step, the solver extends A by making a decision literal l_d at decision level $d + 1$, applies Boolean Constraint Propagation (BCP) to deduce forced assignments and update G, and, if a conflict clause C is found (i.e., all literals are false under A), analyzes G to identify the first Unique Implication Point (UIP). From this analysis, a learned clause C_{learn} is derived to block the conflict and added to F. The solver then backjumps to the highest decision level d' at which C_{learn} becomes unit and continues the search from level d'.

A *Unique Implication Point* (UIP) in the implication graph G is a node at the current decision level d such that every path from the decision literal l_d to the conflict node passes through it. Intuitively, a UIP acts as a critical cut-point that uniquely separates the initial decision assignment from the conflict.

The *First UIP* is the UIP closest to the conflict node along all paths in G. Identifying the First UIP is crucial because it enables the solver to derive a *learned clause* C_{learn} that succinctly captures the cause of the conflict. This learned clause prevents the solver from repeating the same conflicting assignments, effectively pruning the search space.

Moreover, using the First UIP allows the solver to perform *non-chronological backtracking* (or backjumping), where the solver backtracks directly to the highest decision level $d' < d$ at which C_{learn} becomes unit. This targeted backjumping avoids reconsidering irrelevant decision levels, greatly improving the efficiency of the search.

Thus, by learning from conflicts and backjumping non-chronologically, CDCL solvers drastically reduce redundant search and efficiently solve large, complex SAT instances. This approach forms the basis of most state-of-the-art SAT and SMT solvers.

It is worth noting that SMT natively supports richer theories compared to CP, including infinite or large domains such as integers, reals, and complex data structures, and is therefore not limited to finite domains. Despite these differences, in some cases it is possible to model the same problem using either a CP or an SMT formulation.

Chapter 3

Related Works

In the previous chapter, we introduced the theoretical and methodological background necessary to understand the concepts and techniques used throughout this work. In this chapter, instead, we focus on the approaches that are most directly comparable to our study. Specifically, we provide an overview of the existing literature on anomaly detection in railway systems, highlighting the main methodologies, data sources, and modeling strategies proposed in this domain. As we will see, the methods developed to identify anomalies are numerous and span fields ranging from traditional Machine Learning to Deep Learning techniques.

Anomaly Detection in Railways This topic is widely studied, and many papers are present in the literature. A large number of them focus on finding anomalies in railway systems using images as the data source. However, it is not always possible to work with images. In particular, a valuable data source is represented by sensors installed on board.

Origlia et al. [39] worked with wheel accelerometric sensors to detect anomalies on the track. In their work, the authors sampled data from the field using a 400 m long straight railway section as trackside. Each train trip created a different univariate temporal series. They also used two different types of trains with different speeds. In this case, the train maintained a constant speed during the trip. Note also that the sampling period was fixed and did not vary. As stated by the authors, the scope of this work was to use a univariate series in order to make a comparison between different types of anomaly detection models, leaving multivariate analysis to future works.

In this work, the authors tested three types of Deep Learning methods: an Autoencoder, an LSTM [20] for anomaly detection (LSTM-AD) [33], and an LSTM model based on an Encoder-Decoder architecture (LSTM-ED) [32]. In particular, they decided to use the LSTM-AD model to find anomalies because they had a time series dataset; the idea is that it could be important to take into account the temporal dependencies in data, which is not possible using an autoencoder. Note that, differently from the autoencoder, the LSTM-AD uses a prediction error and not a reconstruction error to find anomalies. Meanwhile, in the LSTM-ED, the reconstruction error is used to train the model, and the same signal used for the LSTM-AD is used to find anomalies.

To compare these three models, the authors decided to avoid thresholding mechanisms because the performances are extremely dependent on these mechanisms. Thus, they used the ROC curve and AUROC. In their experiments, the authors highlighted how the LSTM-AD slightly outperformed other methods but required longer computation time. In particular, the LSTM-AD obtained an AUROC of 0.83, the AE model performed at 0.81, while the LSTM-ED presented a performance of 0.80. Thus, taking into account the temporal dependencies in data was useful in this use case. Note that the authors highlighted how these results can be influenced by the fact that no external factor, such as variation in train velocity, affected the time series.

It is not always possible to work with only univariate time series; in some real domains, it is required to consider all the signals derived from sensors in the field, moving the analysis into the multivariate realm. However, in data-driven domains such as railway signaling, it can be important to reduce the number of features to speed up the models and the overall system. Indeed, some features may be redundant and thus can be eliminated. Xu et al. [66]

proposed an unsupervised feature extraction method in the high-speed train domain. In this work, the authors had a series of signals derived from sensors related to the brake system of high-speed trains. Their goal was to find anomalies in this multivariate domain to identify critical situations along the line.

To find these anomalies, they decided to use two machine learning approaches: Support Vector Machines (SVM) and the Random Forest (RF) classifier. In this case, temporal dependencies in data were not taken into account, but the dataset was multivariate. Since the number of features was high, they performed feature reduction to speed up the models and remove useless information. Their aim was to select only variables with a high causal effect on faults; thus, they performed a ”causality-based feature extraction” based on building a causal network using the BIC scoring function and hill-climbing algorithm. Once the network was built, they identified nodes that had no children. These nodes are not causal of anything and were therefore eliminated. This method was also compared with other methods such as Principal Component Analysis (PCA) and Correlation Ranking (CR). The proposed feature extraction method outperformed both PCA and CR.

Islam et al. [22] introduced a novel anomaly detection system for the Internet of Railways (IoR) using extended neural networks. In this work, the authors used two different open-source datasets, focusing on multivariate anomaly detection. The goal was to identify anomalies in multivariate signals to detect possible network attacks by hackers on the infrastructure network. The authors selected features using a k-means clustering technique. In particular, they performed a ranking on each feature based on the distance between that specific feature and the centroid of the cluster; this was done for each feature and each point to assess how similar the feature is to the specific point.

To find anomalies, they did not rely on machine learning approaches but on a deep learning one. Specifically, they used a neural network to find anomalies for each point. However, as highlighted by the authors, in some cases it could be useful to take into account the temporal dependencies in data. Thus, they extended the neural network with additional parameters to be tuned: a scaling parameter (μ) and a shifting parameter (γ). By tuning these two parameters, the authors tried to map variations inside the data, partially accounting for temporal information.

In some cases, it is possible to find anomalies in data without using deep learning approaches, but rather using machine learning methods. To handle vibration signals in railways, Zou et al. [70] proposed using the Isolation Forest model to detect anomalies, combined with PCA and Laplacian score-based feature ranking for feature selection.

In this work, the authors needed to find squats in switches and crossings along the railway. They could not use sensors along the line, so they decided to work with an accelerometer mounted on board. The idea behind this approach is that studying the vibration signals of the train when it passes through a squat allows automatic detection, because the vibration is the result of a dynamic response to the wheel–rail interaction. To simulate a real-world application, the authors built a testbed with a short railway track comprising two different squats with two different intensities. Then, they mounted the accelerometer and gathered data from the test track.

From the vibration signal, they obtained eleven different features such as skewness, peak-to-peak amplitude, and standard deviation. Again, in this work, as in previous ones, the train velocity was constant, as was the sensor sampling period.

As introduced before, the authors performed feature selection based on a score value. To do this, they used two methods: the first based on PCA, and the second based on a Laplacian feature selection procedure. In particular, they computed the score of all features, then found the most important one. They computed the cross-correlation between the most important feature and all others, removing those with higher correlation to the most important feature. This procedure was repeated until no two features had a cross-correlation higher than 0.9.

To find anomalies in the signal, they used Isolation Forest, building an anomaly score for each point. Since an anomaly score was given, they needed a threshold. They ordered these scores in descending order and found the knee point of this function, which represented their threshold.

The results showed that Isolation Forest was able to detect almost all anomalies on the track in an unsupervised manner. It is worth noting that the built dataset was not labeled, but the authors knew a priori the position of anomalies because the track was artificially constructed.

Contamination Factor Estimation One of the main limitations of the Isolation Forest algorithm when used in an unsupervised setting lies in its dependency on a hyperparameter called the *Contamination Factor*, which represents the expected proportion of anomalies within the dataset, as stated by the authors of the original paper [30]. The primary issue is that, in most real-world scenarios, it is not possible to know in advance the exact percentage of anomalous data points, and estimating this ratio can be a challenging task. A possible estimation approach was proposed by Perini et al. [44], who introduced a Bayesian method called *GammaGMM* to estimate the contamination factor's posterior distribution. The method relies on a pipeline that combines the outputs of multiple unsupervised anomaly detection algorithms to infer the prior probability of the contamination factor. The overall process is divided into four main steps. In the first step, an *anomaly space* is constructed: since each anomaly detection algorithm produces an anomaly score, each data point is represented as a vector of anomaly scores, one per detector. In the second step, the resulting points in this new space are modeled using a Dirichlet Process Gaussian Mixture Model (DPGMM), producing a mixture of components. The third step consists of ordering these mixture components from the most to the least anomalous and estimating the probability that the first k components correspond to anomalous data. Finally, in the fourth step, GammaGMM estimates the posterior distribution of the contamination factor based on these probabilities and the components' mixing proportions. However, the main limitation of this method is that, to estimate the contamination factor, the authors introduced two additional hyperparameters that require supervision. Specifically, the first hyperparameter represents the prior belief about the presence of anomalies in the data, while the second reflects how probable it is that a large number of anomalies are present. This design choice stems from the authors' intention to characterize the *uncertainty* of the contamination factor rather than determining a single deterministic value. GammaGMM was evaluated on 22 benchmark datasets, successfully estimating the contamination factor for Isolation Forest in the majority of cases, demonstrating its robustness and general applicability in unsupervised anomaly detection scenarios.

Anomaly Detection in Radio Signals In the field of radio signals, addressing electromagnetic interference in railway communication, an important contribution was made by Fan et al. [11]. In this paper, the authors introduced a real-time anomaly detection system based on Autoencoders and LSTM to identify anomalies in multivariate 5G signaling time series. They specifically studied the models' capacity to handle long-term dependencies in time series.

The goal of this work was to find anomalies in the electromagnetic field, dealing with radio coverage between the base station and the train. They worked with a multivariate dataset gathered from a real application and tested the system in real time. To find anomalies, the authors used an AE-LSTM model trained with mean squared error (MSE) as the loss function. This approach aimed to handle temporal dependencies in data while leveraging the power of an autoencoder-based architecture.

They compared the model with several others, such as the autoencoder, a convolutional neural network-based autoencoder, and support vector machines. They obtained the best accuracy results with the AE-LSTM, outperforming other methods by 0.7%. On the other hand, in terms of online detection time, this model was the slowest, requiring roughly twice the time needed by other methods to detect anomalies.

As noted by the authors, the main limitation is that the entire dataset focused solely on one frequency domain, neglecting frequency variance over time. Thus, with different frequencies, the results could degrade because the AE-LSTM is not invariant to changes in frequency or sampling time in the dataset. In particular, they highlighted how varying train speed could also reduce accuracy.

Synthetic Data Generation For what concerns the synthetic data generation task, an interesting contribution was presented by Coletta et al. [8], who introduced several methods for constraint-aware synthetic data generation. As they highlighted, most existing SDG approaches are unable to generate datasets or time series that satisfy inter-feature or temporal constraints. In their work, the authors proposed a diffusion-based model called *DiffTime* for generating synthetic time series data. They also introduced a guided version of this model, named *GuidedDiffTime*, which is a conditional diffusion model that does not require retraining when new constraints are introduced into the data and demands a reduced number of training steps.

Another method proposed for synthetic data generation is what the authors refer to as the *Constraint Optimization Problem (COP)* approach. In this framework, the task of synthetic data generation is formulated as a constraint optimization problem, where the constraints are defined either across time steps or among features within the same record. Specifically, the authors first generate synthetic data using an SDG model and then fine-tune each record through COP. In practice, for each generated record, COP finds the optimal solution that minimizes the L2 distance between the generated sample and the target objective. In this way, the authors demonstrated that it is possible to preserve temporal consistency by adjusting nearly correct samples instead of discarding them through rejection sampling. Their experiments showed that both COP and DiffTime effectively produce synthetic time series that share the same latent space as real data, while also requiring relatively low training time and computational resources.

In conclusion, the reviewed studies emphasize the importance and potential of Artificial Intelligence approaches for anomaly detection in the railway systems domain. In particular, there is a noticeable trend toward the use of autoencoder-based architectures and LSTM models to effectively handle temporal data. Nevertheless, traditional machine learning methods also remain viable alternatives. Additionally, feature selection plays a critical role in many studies, helping to improve computational efficiency and reduce data redundancy. Finally, an innovative use of Constraint Programming has been proposed for synthetic data generation, allowing the fine-tuning of generated samples under structural constraints.

Chapter 4

Data

The data used in this work were entirely collected from the field, thus fully representing a real-world application. Specifically, multiple datasets were constructed due to differing data collection conditions across various scenarios. These datasets were created by merging two distinct data sources originating from two different programs. The final outcome is a dataset describing the communication quality between the onboard radio modem and the backbone network with which the train was communicating.

The datasets are essentially Comma Separated Values (CSV) files, organized by collection day, backbone, and communication stream.

The datasets comprise a total of 25 features:

- **Pair**: an integer value representing the group identifier used by experts during data collection; essentially a configuration parameter;
- **Record Number**: a monotonically increasing integer indicating the record's position within the dataset;
- **Transaction Count**: an integer representing the number of transactions that occurred between the previous record and the current one;
- **Bytes Sent by E1**: an integer indicating the number of bytes sent by the first endpoint, which corresponds to the backbone WiFi radio modem

when analyzing ground-to-train communication, or the onboard radio modem when analyzing train-to-ground communication;

- **Bytes Rcvd by E2 (Streaming)**: an integer indicating the number of bytes received by the second endpoint, which corresponds to the on-board radio modem in ground-to-train communication, or the backbone WiFi radio modem in train-to-ground communication;

- **Bytes Lost from E1 to E2**: an integer representing the number of bytes lost during communication between the two endpoints;

- **Inactive Time (sec)**: an integer representing the number of seconds during which the recording program was inactive;

- **Throughput (Mbps)**: a floating-point value indicating the number of megabits per second transmitted by the first endpoint during communication;

- **RFC 1889 Jitter (ms)**: an integer measuring jitter in milliseconds, calculated according to RFC 1889, which applies a continuous average of individual packet jitter smoothed by a factor of 16;

- **One-Way Delay (ms)**: an integer representing the total time, in milliseconds, taken by a single datagram to travel from endpoint one to endpoint two;

- **Jitter (delay variation) Maximum (ms)**: an integer indicating the maximum jitter (delay variation) observed since the previous record, expressed in milliseconds;

- **Jitter (delay variation): 0-10 (ms)**: an integer counting how many times the jitter ranged between 0 and 10 milliseconds since the previous record;

- **Jitter (delay variation): 11-20 (ms)**: an integer counting how many times the jitter ranged between 11 and 20 milliseconds since the previous record;
- **Jitter (delay variation): 21-30 (ms)**: an integer counting how many times the jitter ranged between 21 and 30 milliseconds since the previous record;
- **Jitter (delay variation): 31-50 (ms)**: an integer counting how many times the jitter ranged between 31 and 50 milliseconds since the previous record;
- **Jitter (delay variation): 51+ (ms)**: an integer counting how many times the jitter exceeded 51 milliseconds since the previous record;
- **Maximum Consecutive Lost Datagrams**: an integer representing the maximum number of consecutive lost datagrams since the previous record;
- **Consecutive Lost Datagrams: 1**: an integer counting occurrences of a single consecutive lost datagram during communication;
- **Consecutive Lost Datagrams: 2-3**: an integer counting occurrences of two or three consecutive lost datagrams during communication;
- **Consecutive Lost Datagrams: 4-5**: an integer counting occurrences of four or five consecutive lost datagrams during communication;
- **Consecutive Lost Datagrams: 6-10**: an integer counting occurrences of between six and ten consecutive lost datagrams during communication;
- **Consecutive Lost Datagrams: 11+**: an integer counting occurrences of eleven or more consecutive lost datagrams during communication;
- **Timestamp**: the timestamp associated with the record;

- **Mac Address**: a string representing the MAC address of the WiFi radio modem to which the train was connected;
- **RSSI**: an integer representing the Received Signal Strength Indicator sampled by the onboard radio modem.

4.1 Data Gathering from Field and Dataset Building

As introduced previously, each data sample used in this work was collected in the field, thus fully representing data from a real-world application.

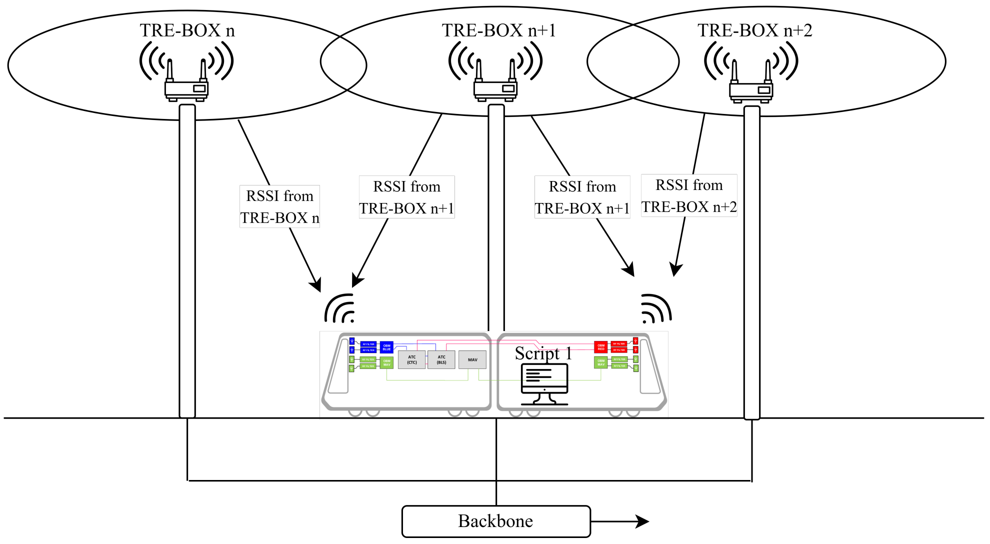


Figure 4.1: Schematic representation of the data acquisition process in the field. A first script runs onboard the train, collecting RSSI values from each detectable TRE-BOX. Simultaneously, a second script gathers information from the backbone network regarding the quality of the end-to-end communication between the train and the ground radio access points.

Data collection was performed using two different programs (see Figure 4.1), each serving distinct purposes. To gather data, a test train was equipped with a computer running a Linux distribution, connected to the onboard radio modems (OBMs). Before starting the train trip, the OBMs were configured to connect to a specific backbone network (RED, BLUE, or GREEN). Upon departure, the OBMs connected to the first available WiFi modem within the selected backbone. A script running on the onboard computer recorded, at predefined sampling intervals, the RSSI value sensed by the onboard radio modem at each instant. For every timestamp, the script logged the MAC address of the connected WiFi modem on the backbone along with the corresponding RSSI value. Additionally, the script recorded the moments when a hand-off occurred; which is defined as the event when the onboard radio modem disconnects from one WiFi modem and connects to the next.

In parallel, a second script collected information from the backbone to monitor the quality of the connection between the two endpoints (the onboard radio modem and the backbone). Specifically, this script gathered metrics such as the number of bytes sent, received, lost, and other relevant parameters. Essentially, it collected all features of the dataset except for the timestamp and RSSI. The sampling interval of this script was user-configurable and not necessarily constant.

A primary challenge was that the two scripts operated with unsynchronized sampling intervals. Therefore, a synchronization step was necessary to construct the final dataset, along with a dedicated dataset building procedure. Given the initial timestamp recorded by the second script, it was possible to align each record from the second script with the corresponding data from the first script.

Data Synchronization To synchronize the data collected by the two scripts, for each record sampled by the second script we appended the RSSI value recorded by the first script that was temporally closest to that record.

More specifically, let us denote by $\{t_i^{(2)}\}_{i=1}^N$ the timestamps of the records collected by the second script, and by $\{(t_j^{(1)}, \mathrm{RSSI}_j, \mathrm{MAC}_j)\}_{j=1}^M$ the tuples of timestamps, RSSI values, and MAC addresses collected by the first script. For each timestamp $t_i^{(2)}$, we first select only those records from the first script whose MAC address MAC_j matches the MAC address connected to the on-board radio modem (OBM) at time $t_i^{(2)}$.

Then, we find the index j^* such that the temporal distance $|t_i^{(2)} - t_j^{(1)}|$ is minimized:

$$j^* = \arg \min_{\substack{j=1,\ldots,M \\ \mathrm{MAC}_j=\text{MAC at } t_i^{(2)}}} |t_i^{(2)} - t_j^{(1)}|$$

The RSSI value RSSI_{j^*} corresponding to this index is then associated with the record at time $t_i^{(2)}$.

This procedure ensures that each record from the second script is enriched with the RSSI measurement that is most temporally relevant and corresponds to the same connected WiFi modem, thus effectively synchronizing the two datasets despite their asynchronous sampling intervals.

Note that the first script generated textual log files, while the second produced comma-separated value (CSV) files. To build the dataset, the log files were processed using regular expressions to extract RSSI values, timestamps, MAC addresses, and hand-off events. Then, for each record sampled by the second script, the MAC address of the WiFi modem connected to the OBM, along with the corresponding timestamp and RSSI value extracted from the first script, were appended.

4.2 Expert Labelling

One of the main challenges when dealing with artificial intelligence approaches is the need to evaluate model performance using appropriate metrics. In our case study, this requires prior knowledge of the location of anomalies within the datasets; in other words, it is necessary to know which records are anomalous and which are not.

Defining what constitutes an anomaly in these radio signals was not straightforward, even for domain experts. By ”domain experts” we refer here to the telecommunications and network engineers who collected the field data and whose role is to design and build the radio network for the railway infrastructure. To identify anomalies in the radio signals, the experts needed to analyze the signal trends over time alongside the train's position. Knowing the train's location along the railway line is crucial because it allows understanding whether disturbances in the radio signal are caused by physical infrastructure characteristics. For example, tunnels and curves typically introduce more signal interference, while minor disturbances during hand-offs are not necessarily indicative of coverage anomalies.

For the purpose of this work, it was therefore necessary to label some datasets manually. To facilitate this, we developed a script capable of plotting multiple features over time simultaneously, highlighting the occurrence of hand-offs and the associated MAC addresses. This allowed the experts to infer the train's position during the trip, since each MAC address is associated with a known train station.

The expert labelling process consisted of carefully reviewing signal data from different acquisitions by two independent domain experts. These labelling sessions were conducted separately to prevent any bias or influence between the experts. During the sessions, the experts identified time windows containing anomalous records by visually inspecting the radio signal plots.

It is important to note that in the majority of cases, the experts agreed on the labelling, but some discrepancies did arise. To account for this, multiple labelled versions of the same dataset were created. Consequently, after data acquisition, dataset building, and expert labelling, we obtained several datasets from all three backbones (RED, BLUE, and GREEN), collected under different transmission settings and annotated with varying labels reflecting expert agreement and disagreement.

Chapter 5

Method

This chapter describes the methodology adopted throughout the study, detailing the main phases from the investigation of synthetic data generation methods to the anomaly detection process. To guide the reader, the overall workflow of the entire work, including data collection and software implementation, is summarized in Figure 5.1.

The process begins with the collection of real data and their subsequent expert labeling, presented in detail in chapter 4. Afterwards, several synthetic data generation methods were tested, and the most effective one was selected to build a synthetic data generation pipeline, which was then used to create datasets combining both real and synthetic samples.

Using these datasets, multiple experimental analyses were conducted to identify the best strategy for estimating the contamination factor in Isolation Forest, to compare different thresholding methods, and to study how varying proportions of real and synthetic data influence anomaly detection performance. The anomaly detection methods were then evaluated on several datasets, containing both real and synthetic data, and further tested with different feature selection methods previously analyzed.

Finally, two simple software tools (which are presented in Appendix C) were developed: one for building the datasets and another for detecting anomalies using pretrained anomaly detection models.

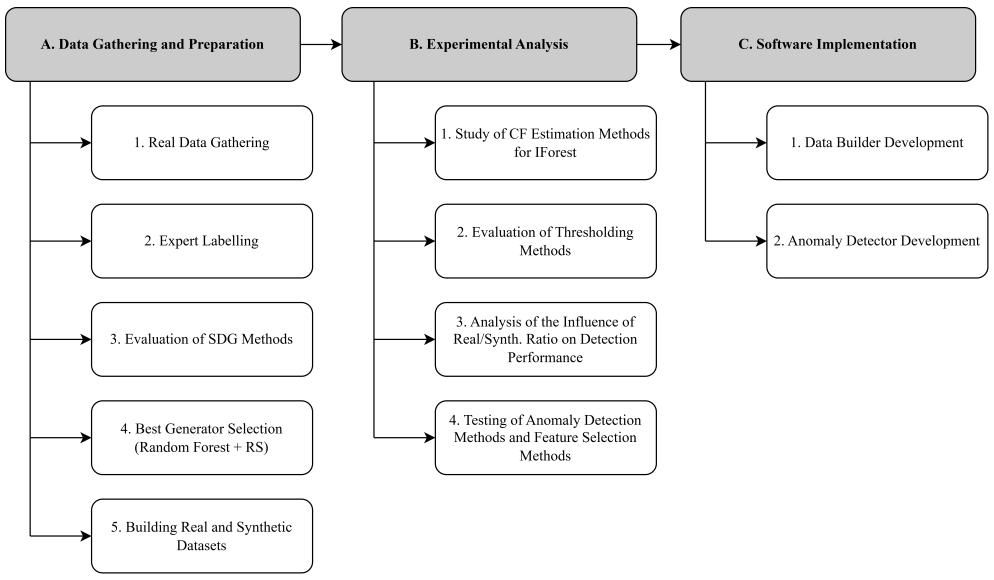

A. Data Gathering and Preparation
1. Real Data Gathering
2. Expert Labelling
3. Evaluation of SDG Methods
4. Best Generator Selection (Random Forest + RS)
5. Building Real and Synthetic Datasets
B. Experimental Analysis
1. Study of CF Estimation Methods for IForest
2. Evaluation of Thresholding Methods
3. Analysis of the Influence of Real/Synth. Ratio on Detection Performance
4. Testing of Anomaly Detection Methods and Feature Selection Methods
C. Software Implementation
1. Data Builder Development
2. Anomaly Detector Development

Figure 5.1: Overall workflow of the proposed methodology, from data collection and synthetic data generation to anomaly detection experiments and software implementation.

5.1 Synthetic Data Generation and Evaluation

A major challenge encountered during this work was the scarcity of data containing few anomalies. As previously mentioned, the radio infrastructure was not fully completed, resulting in imperfect radio coverage. Consequently, the occurrence of anomalous data was extremely high, and the amount of anomaly-free data risked being insufficient to effectively train AI-based anomaly detection methods. To address this issue, we sought a way to generate high-quality synthetic data free of anomalies. Accordingly, we selected several synthetic data generation (SDG) methods and evaluated their performance to identify the most suitable one.

Specifically, we applied different types of SDG methods and various evaluation metrics with the goal of selecting the best-performing approach. Regardless of the SDG method used, the procedure was the same: first, we created a smaller dataset derived from real data, containing only records that experts had identified as non-anomalous. Each SDG method was then trained on this clean dataset to generate a new synthetic dataset. Finally, we conducted a series of evaluation steps to assess the quality of the synthetic data produced.

The SDG methods employed in this study include *SMOTE*, *DTWBA*, *GMM*, *ForestDiffusion*, *DoppleGANger*, *CTGAN*, *TVAE*, *PAR*, *Chronos*, and *TimesFM*. We chose these methods to cover a broad spectrum of categories within the SDG domain. This selection allowed us to test deterministic approaches such as DTWBA and SMOTE, as well as machine learning and deep learning techniques, including transformer-based models.

On the other hand, to assess the quality of the synthetic data, we employed various metrics and techniques. First, we visually inspected the overlap between the distributions of the synthetic and real data. Additionally, we examined whether the two datasets shared a similar latent space by analyzing 2D embeddings generated using both *t-SNE* and *UMAP*. To obtain quantitative measures of quality, we computed several statistical distances, including the *Maximum Mean Discrepancy (MMD)*, the *Earth Mover's Distance (EMD)*, and *Dynamic Time Warping (DTW)*. Finally, to evaluate how closely the synthetic data resembled the real data, we conducted a two-sample discrimination test using a *Random Forest* classifier trained to distinguish between real and synthetic samples.

An important limitation of some SDG methods is their inability to guarantee or learn the constraints between features in the synthetic data. For example, if a feature F_1 is related to another feature F_2 by a constraint C, denoted as $C(F_1, F_2)$, the synthetic data generated by certain methods may violate this constraint. Such violations can significantly degrade the quality of the synthetic dataset. Some methods, however, are capable of learning these constraints due to their architectural design. Alternatively, one can generate synthetic data without enforcing constraints initially, and then find the closest possible records that satisfy the constraints to replace the original ones. To achieve this, we formulated the problem as a constraint optimization task and employed an SMT solver to find suitable solutions.

Using an SMT Solver to Refine Synthetic Data Since it is not always possible to train generative models that inherently respect feature constraints, one viable approach is to first generate synthetic data using a trained model, then identify and replace any records that violate constraints with the nearest feasible records that satisfy them. This post-processing step involves substituting the original invalid records with corrected ones. We mapped this problem into a constraint optimization framework and solved it using an SMT solver.

Formally, let us denote the set of features (variables) for a single data record as $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathcal{X}$, where n is the number of features and $\mathcal{X}$ is the domain of possible feature values. The features must satisfy a set of constraints expressed as logical formulas or inequalities:

$$C(\mathbf{x}) = \{c_1(\mathbf{x}), c_2(\mathbf{x}), \ldots, c_m(\mathbf{x})\}$$

where each $c_i(\mathbf{x})$ represents a constraint that must hold true.

Given an initial synthetic record $\mathbf{x}^0$ generated by a model, the goal is to find a corrected record $\mathbf{x}^*$ that satisfies all constraints:

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathcal{X}} \; d(\mathbf{x}, \mathbf{x}^0) \quad \text{subject to} \quad \bigwedge_{i=1}^{m} c_i(\mathbf{x})$$

where $d(\cdot, \cdot)$ is a distance function measuring the difference between the original and corrected feature values.

The SMT solver searches for an assignment $\mathbf{x}^*$ that satisfies the constraints while minimizing the distance $d(\mathbf{x}, \mathbf{x}^0)$, thus producing synthetic data records that are both valid and as close as possible to the original generated data.

The first step consisted in dropping all the features in the data without a relation with other features or those that can be computed manually using the values of the others. Thus, we dropped the following features: *Pair*, *Record Number*, *Transaction Count*, *Bytes Lost from E1 to E2*, *Inactive Time (sec)*, *Throughput (Mbps)*, *RFC 1889 Jitter (ms)*, *One-Way Delay (ms)*, *Timestamp*, *Mac Address*, and *RSSI*. In particular, *Throughput (Mbps)*, *RFC 1889 Jitter* *(ms)*, *One-Way Delay (ms)*, and *RSSI* are features not directly related to others by constraints; thus, the generated values for these variables are acceptable independently of the other features. On the other hand, we removed *Bytes Lost from E1 to E2* to reduce the number of variables, since it can be computed afterwards using the values of *Bytes Sent by E1* and *Bytes Rcvd by E2 (Streaming)*. Finally, for *Timestamp* and *Mac Address* we decided to simply copy the real values because these features are not used by the anomaly detection models to solve the task, so their values are not important. Thus, the remaining features were indexed as follows:

- x_1: *Bytes Sent by E1*;
- x_2: *Bytes Rcvd by E2 (Streaming)*;
- x_3: *Jitter (delay variation) Maximum (ms)*;
- x_4: *Jitter (delay variation): 0-10 (ms)*;
- x_5: *Jitter (delay variation): 11-20 (ms)*;
- x_6: *Jitter (delay variation): 21-30 (ms)*;
- x_7: *Jitter (delay variation): 31-50 (ms)*;
- x_8: *Jitter (delay variation): 51+ (ms)*;
- x_9: *Maximum Consecutive Lost Datagrams*;
- x_{10}: *Consecutive Lost Datagrams: 1*;
- x_{11}: *Consecutive Lost Datagrams: 2-3*;
- x_{12}: *Consecutive Lost Datagrams: 4-5*;
- x_{13}: *Consecutive Lost Datagrams: 6-10*;
- x_{14}: *Consecutive Lost Datagrams: 11+*.

The subsequent steps involved defining, for each of the remaining feature, three constant values: the minimum value of the feature observed in the real data, denoted as $\min_i$; the maximum value of the feature in the real data, denoted as $\max_i$; and the value generated by the synthetic data for that feature, denoted as s_i, where $i = 1, \ldots, n$ indexes the features. For each feature i, a decision variable x_i was introduced to represent the refined value of that feature. The domain of each decision variable was constrained by the observed minimum and maximum values, enforcing the bounds:

$$\min_i \leq x_i \leq \max_i, \quad \forall i = 1, \ldots, n$$

This ensures that the solution found by the SMT solver respects the realistic range of each feature based on the original dataset.

Then, we defined a series of constraint between features:

1. The number of sent bytes must be a multiple of 4:

$$x_1 \equiv 0 \pmod{4}$$

2. The number of received bytes must be a multiple of 4:

$$x_2 \equiv 0 \pmod{4}$$

3. The number of sent bytes must be greater than or equal to the number of received bytes:

$$x_1 \geq x_2$$

4. If the difference between sent and received bytes is greater than zero, then the maximum number of lost datagrams must be greater than zero:

$$(x_1 - x_2 > 0) \implies (x_9 > 0)$$

5. If the maximum number of consecutive lost datagrams equals 1, then *Consecutive Lost Datagrams: 1* must be greater than zero, and all other lost datagrams features must be zero:

$$(x_9 = 1) \implies (x_{10} > 0 \land x_{11} = 0 \land x_{12} = 0 \land x_{13} = 0 \land x_{14} = 0)$$

6. If the maximum number of consecutive lost datagrams is between 2 and 3 inclusive, then *Consecutive Lost Datagrams: 2-3* must be greater than zero, and all higher categories must be zero:

$$(2 \leq x_9 \leq 3) \implies (x_{11} > 0 \land x_{12} = 0 \land x_{13} = 0 \land x_{14} = 0)$$

7. If the maximum number of consecutive lost datagrams is between 4 and 5 inclusive, then *Consecutive Lost Datagrams: 4-5* must be greater than zero, and all higher categories must be zero:

$$(4 \leq x_9 \leq 5) \implies (x_{12} > 0 \land x_{13} = 0 \land x_{14} = 0)$$

8. If the maximum number of consecutive lost datagrams is between 6 and 10 inclusive, then *Consecutive Lost Datagrams: 6-10* must be greater than zero, and the highest category must be zero:

$$(6 \leq x_9 \leq 10) \implies (x_{13} > 0 \land x_{14} = 0)$$

9. If the maximum number of consecutive lost datagrams is greater than or equal to 11, then *Consecutive Lost Datagrams: 11+* must be greater than zero:

$$(x_9 \geq 11) \implies (x_{14} > 0)$$

10. If the maximum jitter is between 0 and 10 inclusive, then *Jitter (delay variation): 0-10 (ms)* must be greater than zero, and all higher jitter

categories must be zero:

$$(0 \leq x_3 \leq 10) \implies (x_4 > 0 \land x_5 = 0 \land x_6 = 0 \land x_7 = 0 \land x_8 = 0)$$

11. If the maximum jitter is between 11 and 20 inclusive, then *Jitter (delay variation): 11-20 (ms)* must be greater than zero, and all higher jitter categories must be zero:

$$(11 \leq x_3 \leq 20) \implies (x_5 > 0 \land x_6 = 0 \land x_7 = 0 \land x_8 = 0)$$

12. If the maximum jitter is between 21 and 30 inclusive, then *Jitter (delay variation): 21-30 (ms)* must be greater than zero, and all higher jitter categories must be zero:

$$(21 \leq x_3 \leq 30) \implies (x_6 > 0 \land x_7 = 0 \land x_8 = 0)$$

13. If the maximum jitter is between 31 and 50 inclusive, then *Jitter (delay variation): 31-50 (ms)* must be greater than zero, and the highest jitter category must be zero:

$$(31 \leq x_3 \leq 50) \implies (x_7 > 0 \land x_8 = 0)$$

14. If the maximum jitter is greater than or equal to 51, then *Jitter (delay variation): 51+ (ms)* must be greater than zero:

$$(x_3 \geq 51) \implies (x_8 > 0)$$

Then, we required to define an objective function to minimize. In order to do that, we decided to use a distance metric able to take into account the distance feature by feature between the synthetic values and the assignments done by the SMT solver on the decision variables. Thus, we decided to use the

Mean Absolute Error (MAE). We selected this as objective function in order to avoid quadratic constraints such as in the case of a Mean Squared Error (MSE), which can be more complex and computationally expensive to solve in the SMT framework. The MAE is defined as the average of the absolute differences between predicted and actual values across all features. Formally, for features indexed by $i = 1, \dots, n$, it is given by:

$$\mathrm{MAE} = \frac{1}{n} \sum_{i=1}^{n} |x_i - \mathbf{x}_i^0|$$

where x_i are the values assigned by the solver, and $\mathbf{x}_i^0$ are the generated synthetic values.

Since SMT solvers generally do not provide a built-in absolute value function, we implemented the absolute value using the conditional expression operator ITE (if-then-else). For each difference $d_i = x_i - \mathbf{x}_i^0$, the absolute value is expressed as:

$$|d_i| = \mathrm{ITE}(d_i > 0, d_i, -d_i)$$

This means: if the difference d_i is greater than zero, then its absolute value is d_i itself; otherwise, it is the negation $-d_i$.

In the implementation, for each feature, we compute the difference between the original and synthetic integer values, then apply the ITE construct to obtain the absolute difference. Finally, we sum all absolute differences and divide by the number of features to get the MAE, which is minimized by the solver.

Synthetic Data Generation Pipeline As previously discussed, in this work it was necessary to generate synthetic records in order to construct datasets with a sufficient amount of non-anomalous data for training anomaly detection models. Consequently, several Synthetic Data Generation (SDG) methods were evaluated to identify the most suitable one for our task. After extensive testing, the *Tabular Variational Autoencoder (TVAE)* was selected as the main model, as it consistently outperformed other methods in terms of the evaluated metrics.

Therefore, a dedicated pipeline was developed to generate synthetic datasets from real data. Since one of the primary evaluation criteria was the ability of SDG methods to deceive a Random Forest classifier in distinguishing between real and synthetic data, this classifier was explicitly integrated into the pipeline.

The process is as follows: given a dataset of real records collected from the field, we first select the subset of non-anomalous records (as identified by domain experts). This subset is then used to train the TVAE model, enabling it to learn the distribution of normal data. Once trained, the model generates synthetic non-anomalous samples. Subsequently, a combined dataset is created by merging the original real (non-anomalous) data with the newly generated synthetic samples. Each record is then labeled in a binary fashion as either *real* or *synthetic*. A Random Forest classifier is trained on this dataset to discriminate between the two classes and subsequently tested.

Ideally, synthetic records classified by the Random Forest as *real* can be considered of high quality, as they successfully "fool" the classifier. Hence, the pipeline retains these records while rejecting those correctly identified as synthetic. This process is repeated iteratively until the desired number of synthetic samples is obtained. The structure of this pipeline is illustrated in Figure 5.2.

This pipeline aims to generate synthetic data of high quality by combining the most effective SDG model with a rejection sampling approach driven by a machine learning classifier trained to distinguish between real and synthetic samples. The framework is designed to flexibly produce any desired amount of synthetic data, ensuring an effective data augmentation process.

A second pipeline was also proposed, adopting a rejection sampling approach based not on classifier fooling but on a similarity threshold computed through a distance metric (see Figure 5.3). In particular, since TVAE does not

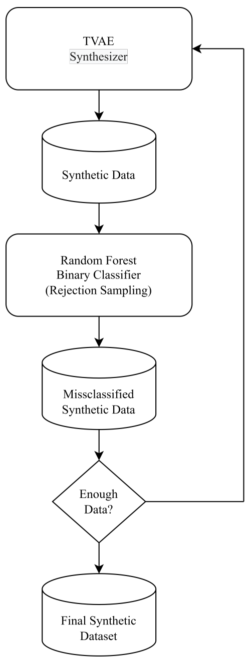


Figure 5.2: Synthetic Data Generation pipeline based on the Tabular Variational Autoencoder and rejection sampling through Random Forest classifier fooling.

preserve temporal dependencies between records, the Dynamic Time Warping (DTW) metric was not employed. Instead, the Maximum Mean Discrepancy (MMD) was used to measure how similar each synthetic record was to the real data in terms of feature-wise value distributions. To determine which samples to retain, a threshold for the distance metric was defined using a percentile-based rule. Specifically, the threshold was set at the 5th percentile of the MMD distances computed between real and synthetic samples.

To evaluate which rejection sampling technique, and consequently which pipeline, performed better, both approaches were tested on three distinct datasets constructed from real data. Specifically, one dataset was created for each backbone: RED, BLUE, and GREEN (the latter representing the multimedia data stream). Each pipeline was applied to all three datasets, and the best-performing one was identified as the pipeline that most effectively reduced the accuracy of the Random Forest classifier in distinguishing between real and synthetic data. In other words, the optimal pipeline was the one capable

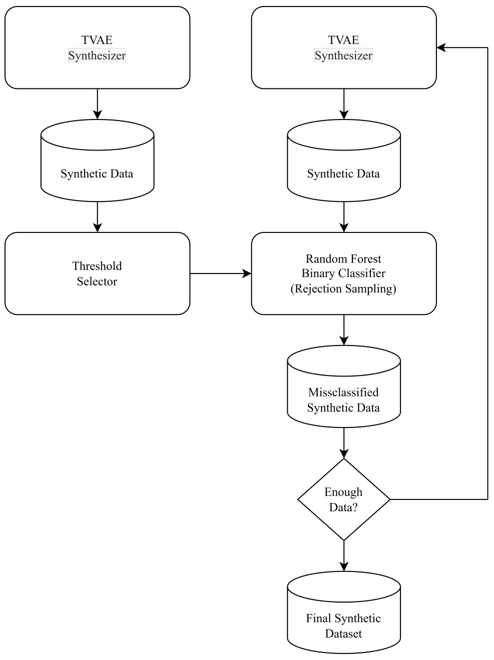


Figure 5.3: Synthetic Data Generation pipeline based on the Tabular Variational Autoencoder and rejection sampling through distance-based thresholding.

of deceiving the classifier to the greatest extent.

5.2 Anomaly Detection

As previously mentioned, the objective of this work was to design a system capable of automatically detecting anomalies in radio coverage signals to assist telecommunication engineers in the implementation of railway radio infrastructures. Therefore, a key component of this work involved selecting and evaluating various anomaly detection methods, while also investigating the impact of synthetic data usage, feature selection, and thresholding mechanisms.

To ensure the robustness of the developed system, multiple anomaly detection methods were employed. Specifically, three approaches were chosen: a machine learning method, *Isolation Forest*; a probabilistic method based on density estimation, *Gaussian Mixture Models (GMM)*; and a deep learning approach, the *Autoencoder*. Each of these methods was tested on several datasets, which included both real and synthetic data, although, in some cases, only real data were used.

Notably, we deliberately chose not to employ anomaly detection models based on temporal dependencies, such as LSTM-based architectures. This decision stems from the nature of the available data: the radio coverage signals were collected under heterogeneous environmental conditions and with varying sampling rates. As a result, temporal continuity between consecutive records cannot be consistently guaranteed across the datasets. Consequently, models that rely on time-series correlations would not be suitable for this context. Instead, the anomaly detection methods adopted in this work were designed to operate under a *temporally independent* assumption, focusing solely on the statistical and structural properties of the data rather than their temporal evolution.

For the *Autoencoder* and *GMM*, which both yield a reconstruction-based output, several thresholding strategies were evaluated to identify the most effective one. Conversely, in the case of the *Isolation Forest*, we investigated different approaches for estimating an important hyperparameter of this method, the contamination factor. Regarding the use of synthetic data, we explored whether it was possible to train anomaly detection models exclusively on synthetic data or to determine the optimal ratio between real and synthetic samples. Finally, a simple feature selection study was carried out to assess whether the dimensionality of the datasets could be reduced without compromising model performance.

5.2.1 Contamination Factor Estimation

During the implementation of the anomaly detection methods, we encountered a practical issue related to the *Isolation Forest* model: it requires the specification of a hyperparameter called the *contamination factor*, which represents the expected proportion of anomalous samples in the data.

However, it is not always possible to have this information *a priori*. Therefore, a reliable method to estimate this hyperparameter was needed in order to employ the Isolation Forest model effectively during the anomaly identification stage. In our case study, domain experts explicitly stated that defining a precise value for this hyperparameter is extremely challenging due to the inherent uncertainties of field data collection.

To address this issue, we decided to design and test several estimation methods in order to identify the most suitable one for our scenario. The selected method would not only be used for the experimental evaluation presented in the previous section but also be implementable in the final anomaly detection pipeline.

In total, we evaluated five different contamination factor estimation methods. Three of them were entirely proposed by us, while the remaining two were taken from existing literature. The following sections describe these methods and the datasets used for their evaluation.

5.2.1.1 Tested Methods

As introduced earlier, several methods were tested to estimate the contamination factor, with the goal of selecting the most appropriate one for the final anomaly detection system. All methods were evaluated using the same performance metrics applied in the anomaly detection evaluation. Specifically, for each test we computed: the *Confusion Matrix*, *False Negative Rate (FNR)*, *Accuracy*, *Precision*, *Recall*, *F1 Score*, and the *Matthews Correlation Coefficient (MCC)*.

All metrics were computed using access points as the reference support rather than individual samples. It is worth noting that three of the proposed methods were specifically designed for this use case.

Automatic Estimation In the original paper on Isolation Forest [30], the authors proposed a simple heuristic for estimating the contamination factor. They tested their model on various datasets under the assumption that the ratio of outliers to normal data is generally very small. Based on this assumption, they fixed the contamination factor to 10%, meaning that one out of ten samples is assumed to be anomalous. Although this assumption is clearly simplistic and unsuitable for many datasets, we decided to include this method as a baseline for comparison.

GammaGMM In this work, we also evaluated the *GammaGMM* method [44], proposed by its authors to estimate the contamination factor for the Isolation Forest model. GammaGMM essentially consists of a pipeline combining multiple anomaly detection algorithms to infer the contamination factor.

In our implementation, the pipeline included the following methods: *K-Nearest Neighbors (KNN)*, *Isolation Forest (IForest)*, *Cluster-Based Local Outlier Factor (CBLOF)*, *Copula-Based Outlier Detection (COPOD)*, *Lightweight Online Detector of Anomalies (LODA)*, *Subspace Outlier Detection (SOD)*, *Local Outlier Factor (LOF)*, *One-Class Support Vector Machine (OCSVM)*, and *Histogram-Based Outlier Score (HBOS)*. The hyperparameters of GammaGMM were kept constant across all tests to ensure a fair comparison.

Domain-based Heuristic on Samples Considering our use case, some anomalies can be systematically identified by analyzing train trips. In particular, it is possible to determine the number of times a deterministic anomaly occurred during a trip. Domain experts indicated that when a deterministic anomaly is detected, an anomaly in the radio signal is also very likely. This is especially true for *self-handoffs*, which occur when the communication between the OBM and the radio access point is unstable, leading the OBM to disconnect and reconnect to the same point.

However, this correlation is not as strong for other anomalies such as *radio jumps* or *color changes*. Considering this partial overlap between deterministic and signal anomalies, we defined a simple heuristic for estimating the contamination factor as:

$$CF = \frac{n_{DA}}{n_{Samples}}$$

where CF is the contamination factor and n_{DA} is the number of deterministic anomalies. The main limitation of this heuristic lies in its sensitivity to dataset size. When the number of samples is large (i.e., the sampling frequency is high), the estimated CF becomes very small, and vice versa. Furthermore, since deterministic anomalies are usually few or even absent, this heuristic often produces very low contamination factor values.

Domain-based Heuristic on TREBOXES To mitigate the previous heuristic's sensitivity to sampling rate, we developed a variant based on the number of radio access points encountered by the train during its trip. In this case, the contamination factor is computed as:

$$CF = \frac{n_{DA}}{n_{TREBOXES}}$$

Here, $n_{TREBOXES}$ represents the number of access points. This heuristic assumes that all samples associated with an access point responsible for a deterministic anomaly are potentially anomalous. Given that the overlap between signal and deterministic anomalies is imperfect, this approach generally leads to a higher contamination factor estimation.

Zero Anomalies Assumption Considering the limitations of the previous approaches, we also tested a method inspired by both our use case and the internal functioning of Isolation Forest. The contamination factor in Isolation

Forest is used to set the thresholds for isolating samples [30], which makes the model highly sensitive to this hyperparameter.

Based on this, we hypothesized that if Isolation Forest is trained with *contamination factor = 0*, it would learn the structure of normal data assuming no anomalies are present. The trained model can then be used to detect anomalies in a separate dataset that actually contains anomalous samples. Although this usage deviates from the traditional design of the model, it offers several advantages: it is simple, comparable across tests (as all methods share the same training set), and not sensitive to dataset size.

5.2.1.2 Performed Tests

To identify the best contamination factor estimation method, we conducted several experiments using real-world data collected in the field. All datasets were labeled by domain experts. Data from all three backbones (*RED*, *BLUE*, and *GREEN*) were included to ensure a fair evaluation. Each test used data collected at different times and under different operational conditions to increase robustness.

For each acquisition, we built two datasets: one containing only normal samples and another including anomalies. The anomalous datasets were used to train and test Isolation Forest with all estimation methods except the *Zero Anomalies Assumption*. For that specific method, the model was trained on the normal dataset and tested on the anomalous one. This allowed a fair comparison while maintaining compatibility with the training assumptions.

Test A This test used data from the RED CBTC backbone. The non-anomalous dataset contained 953 samples, while the anomalous dataset contained 1,748 samples, including 5 radio jumps and 4 color changes (9 deterministic anomalies in total).

Test B This test used data from the GREEN multimedia backbone. The normal dataset contained 64,805 samples, while the anomalous dataset contained 6,986 samples but no deterministic anomalies. This case was used to assess the estimation methods' behavior when deterministic anomalies are absent.

Test C For this test, we used data from the BLUE backbone, which presents a more challenging scenario due to ambiguous anomalies. The normal dataset contained 552 samples, while the anomalous dataset contained 2,877 samples, including 7 radio jumps.

Test D Again, data from the BLUE CBTC backbone were used, but from a different acquisition. The normal dataset contained 1,304 samples, while the anomalous dataset contained 2,459 samples and 6 radio jumps.

Test E Finally, this test used data from the GREEN backbone, collected in a separate acquisition from Test B. Due to a higher sampling rate, the datasets were larger: the normal dataset contained 19,205 samples, while the anomalous dataset contained 11,448 samples with 5 deterministic anomalies (all radio jumps).

5.2.2 Study on Thresholding Methods

In the field of anomaly detection, a crucial aspect for models that produce a continuous output, such as a signal or a reconstruction error (e.g., the *Autoencoder*), is the selection of an appropriate thresholding mechanism. The chosen threshold directly affects the model's ability to correctly distinguish between normal and anomalous samples, and therefore has a strong impact on the overall performance. For this reason, one of the main objectives of this study was to evaluate different thresholding strategies to identify the one that yields the highest performance.

To achieve this goal, we tested the *Autoencoder* using multiple thresholding methods across three datasets, each built from the data collected on the three different backbones. Moreover, we considered two types of reconstruction errors: the *Sum of Squared Errors (SSE)* and the *Mean Absolute Error (MAE)*.

For each backbone, a dataset was created and divided into three parts: a training set, a validation set, and a test set. The *Autoencoder* was trained exclusively on the training set, which contained only non-anomalous samples. The threshold value for each thresholding method was then estimated using the reconstruction errors produced on the validation set, which included both normal and anomalous data. Finally, the complete system (i.e., the *Autoencoder* combined with the thresholding method) was evaluated on the test set, which also contained anomalies.

We tested several thresholding mechanisms, including those introduced in the previous sections: *Z-Score*, *K-Means*, *Density-Based Spatial Clustering of Applications with Noise (DBSCAN)*, *Difference Between the Sets (DBS)*, and *Standard Deviations Inflection Point (SDIP)*. In addition, we used a simple baseline method, the *Percentile* thresholding approach, which defines the threshold based on a specified percentile of the reconstruction error distribution. The Percentile method was tested using different percentile values: 25, 50, 75, 90, 95, 98, 98.5, 99, 99.5, 99.8, 99.9, and 100.

During our experiments, we observed that the *DBSCAN*-based method relies heavily on two hyperparameters, ε and *minPts*, which makes it partially supervised. Since our goal was to evaluate purely unsupervised methods, we implemented a data-driven procedure to automatically infer these parameters. This procedure consists of two main stages: (1) estimating the ε parameter using the *K-Nearest Neighbors* (KNN) approach combined with the *KneeLocator* method, and (2) optimizing *minPts* using the *Silhouette Score*.

In the first step, the distance from each point to its k-th nearest neighbor (with $k = 5$) is computed. These distances are then sorted in ascending order to obtain a *k-distance graph*, where the x-axis represents the ordered points and the y-axis the distance to the 5th nearest neighbor. A noticeable change in slope (the *elbow* or *knee*) in this curve indicates the transition from dense to sparse regions. The position of this knee is automatically detected using the *KneeLocator* algorithm, and the corresponding distance is selected as the optimal ε. Once ε is set, several candidate values for *minPts* are evaluated. For each value, DBSCAN is executed and the resulting clusters are assessed using the *Silhouette Score*, which measures how well each point fits within its assigned cluster compared to other clusters. The value of *minPts* that maximizes this score is selected as the optimal one. This makes the DBSCAN configuration both adaptive to the data distribution and robust to noise.

As previously mentioned, these thresholding methods were evaluated on three datasets and using two reconstruction errors. For the **RED backbone**, the training set consisted of 10,600 samples, the validation set included 1,051 samples (with one anomalous access point), and the test set included 1,050 samples (with four anomalous access points). For the **GREEN backbone**, we used a training set of 30,000 samples, a validation set of 10,000 samples (four anomalous access points), and a test set of 26,000 samples (four anomalous access points). Finally, for the **BLUE backbone**, the training set contained 3,523 records, the validation set 1,303 (with 11 anomalous access points), and the test set 2,351 records (with 12 anomalous access points).

To evaluate the performance of each thresholding method, we computed the following Key Performance Indicators (KPIs): the *Confusion Matrix*, *False Negative Rate (FNR)*, *Accuracy*, *F1 Score*, *Precision*, *Recall*, and the *Matthews Correlation Coefficient (MCC)*. Additionally, for each dataset, we plotted the Receiver Operating Characteristic (ROC) curve and computed the *Area Under the Curve (AUC)* to establish a consistent baseline for comparison.

5.2.3 Study on the Ratio between Real and Synthetic Data

During this study, we aimed to investigate whether it is possible to train an anomaly detection model exclusively on synthetic data while still achieving satisfactory results on real-world datasets. Furthermore, we sought to determine whether constructing a training set that combines both real and synthetic data could enhance the ability of the models to detect anomalies. To this end, we systematically varied the ratio between real and synthetic samples in the training datasets to observe how this proportion affects model performance.

In particular, we tested three anomaly detection methods: *Isolation Forest*, *Gaussian Mixture Models (GMM)*, and the *Autoencoder*, using data collected from the three backbones. From each backbone, we extracted a subset of anomalous data and a subset of normal data. Additionally, for each of these, we generated synthetic datasets using the rejection sampling pipeline based on deceiving a Random Forest classifier.

We then evaluated model performance by constructing several datasets as follows: a validation set and a test set, both composed of real data containing anomalies; a training set composed of a mixture of real and synthetic non-anomalous data; and a small set of non-anomalous data, referred to as the *support dataset*, used as a reference. For each experiment, the anomaly detection models were trained on the training set, the validation set was used to determine the detection threshold for GMM and Autoencoder models through the *Standard Deviations Inflection Point (SDIP)* method (introduced earlier in this work), and finally, performance was assessed on the test set. Additionally, each model was evaluated on the support dataset to verify whether it misclassified any samples that were known a priori to be non-anomalous.

This procedure was repeated for each backbone, for each anomaly detection method, and by varying the ratio between real and synthetic samples in the training data. The goal was to analyze how this ratio influences the models' ability to identify anomalies. Specifically, we gradually reduced the amount of synthetic data in the training set by increments of 10%, starting from a dataset composed entirely of synthetic samples and ending with one containing only real data. Importantly, the overall size of the training set was kept constant to ensure a fair comparison across experiments.

For clarity, the dataset configurations for each backbone are summarized below:

- **RED Backbone:** training set of 1,400 records; validation set of 561 records (including 3 anomalous access points); test set of 520 records (including 2 anomalous access points); support set of 220 records.
- **BLUE Backbone:** training set of 1,500 records; validation set of 750 records (including 6 anomalous access points); test set of 926 records (including 8 anomalous access points); support set of 200 records.
- **GREEN Backbone:** training set of 25,000 records; validation set of 18,000 records (including 4 anomalous access points); test set of 18,000 records (including 4 anomalous access points); support set of 5,000 records.

Finally, the anomaly detection models were evaluated using the following key performance indicators (KPIs): *False Negative Rate (FNR)*, *Precision*, *Recall*, *F1 Score*, and *Matthews Correlation Coefficient (MCC)* on both the validation and test sets. For the support dataset, we measured the number of *False Positives* to identify any misclassifications. As in the other experiments, the evaluation metrics were computed with respect to access points rather than individual records. Moreover, for the Autoencoder, we also computed the mean reconstruction error for each dataset, using the Mean Sum of Squared Errors (MSSE) as the reconstruction error metric.

5.2.4 Test on Real and Synthetic Data

In the context of anomaly detection, it is crucial to rigorously test artificial intelligence models to determine their ability to accurately identify anomalies. Therefore, it was necessary to construct appropriate datasets for training and evaluation. The goal of this work was to develop a stable software system capable of assisting radio infrastructure designers during deployment, providing them with the same level of anomaly identification that would normally require domain expertise.

Table 5.1: Summary of the nine experimental tests designed for anomaly detection evaluation.

Test	Backbone	Train Set Data Type	Total Records	Train	Val	Test	# Anomalies	Notes
1	RED	Real + Synth.	12,102	10,000	1,051	1,051	5	Train set composed by both synth. and real data
2	GREEN	Real	71,791	30,000	10,000	31,791	4	Only real data
3	BLUE	Real	7,194	3,523	1,303	2,368	18	Experts' disagreement (hard test)
4	BLUE	Real	7,194	3,523	1,747	1,924	18	Threshold tuning variant of Test 3
5	BLUE	Real	7,194	2,688	1,955	2,551	10	Only agreed anomalies (simpler test)
6	GREEN	Real	30,654	18,151	2,898	9,605	15	Includes steady-state signals
7	GREEN	Real (Tail)	28,377	21,150	715	6,512	39	Tail position acquisition
8	GREEN	Real	25,192	16,729	2,814	5,649	13	Irregular hand-offs
9	BLUE	Synthetic	37,194	30,000	same as T3	same as T3	18	Trained only on synthetic data

To assess the effectiveness and accuracy of the selected anomaly detection methods, a series of experiments was designed using both real-world data collected by domain experts and synthetic data generated through the rejection sampling pipeline based on the Random Forest classifier, as described in the previous section.

In total, nine experiments were designed, each differing from the others in one or more aspects. These differences may concern the specific backbone from which data were collected, the communication and data transmission settings used, or the presence of both real and synthetic samples. Additionally, variations were introduced in the labeling process, as domain experts sometimes disagreed on which signal segments corresponded to actual anomalies. To provide an overview of all the designed experiments, Table 5.1 summarizes the key characteristics of each test setup, including the data source, dataset size, and anomaly configuration.

A particularly relevant aspect of this work was the identification of *access points* (APs) responsible for weak radio coverage or disturbances. Therefore, the main objective was not to classify individual records as anomalous, but to determine which APs caused anomalies in the signal. Based on the task definition provided by domain experts, performance metrics were computed with respect to the MAC addresses of the access points rather than individual samples. An AP was labeled as anomalous if a coverage anomaly occurred during communication between the train and that specific AP. This approach enables telecommunication engineers to localize spatial regions associated with anomalies and to take corrective action directly in the field.

The three chosen anomaly detection methods (*Isolation Forest*, *Gaussian Mixture Models*, and *Autoencoder*) were applied to all designed tests and evaluated systematically. Notably, we used *Standard Deviation Inflection Point (SDIP)* as thresholding mechanishm on signals produced by both GMM and Autoencoder. The evaluation metrics included the *Confusion Matrix*, *False Negative Rate*, *Accuracy*, *Precision*, *Recall*, *F1 Score*, and *Matthews Correlation Coefficient (MCC)*.

In the following sections, the nine tests are presented. Note that the term *anomaly* refers to anomalous regions in the radio coverage signals, rather than to individual access points themselves.

Test 1 This test used data collected from the RED backbone, representing one of the first field acquisitions. Telecommunication engineers gathered a total of 2702 samples during a complete train journey from *Bengasi* to *Fermi* station. Given the long sampling period and the relatively low sampling frequency, the dataset consisted of only 2702 records. Due to the small number of samples, we selected a subset of non-anomalous records (based on expert labeling) to generate additional synthetic non-anomalous samples. The resulting dataset combined real and synthetic non-anomalous data for training, while anomalous samples were divided between validation and test sets. Experts identified five anomalous regions, located as follows:

- Anomaly n°1: near station *Italia 61*;

- Anomaly n°2: near station *Spezia*;
- Anomaly n°3: near station *XVIII Dicembre*;
- Anomaly n°4: between stations *Paradiso* and *Fermi*;
- Anomaly n°5: near station *Fermi*.

The dataset composition was as follows:

- Train set: 10,000 records, with 600 records from the real dataset and 9,400 from the synthetic one;
- Validation set: with 1,051 records comprising 2 of the anomalies seen above;
- Test set: with 1,051 records with the other 3 anomalies.

Test 2 In the second test, only real data were used, collected from the multimedia (GREEN) backbone. The data acquisition covered a full trip from *Fermi* to *Bengasi*, including communication with all access points along the route. Due to a high sampling rate, a total of 71,791 records were gathered. Experts identified four anomalous regions:

- Anomaly n°1: at the beginning the trip, near the station *Fermi*;
- Anomaly n°2: near the station *Paradiso*;
- Anomaly n°3: between stations *Posta Susa* and *Vinzaglio*;
- Anomaly n°4: immediately after the station *Porta Nuova*.

Then, we created the dataset for this test dividing anomalies between validation and test sets:

- Train set: composed by 30,000 records;
- Validation set: with 10,000 records comprising 2 the anomalies seen above;

- Test set: with 31,791 records with the other 2 anomalies.

Test 3 For the BLUE backbone, two independent data acquisitions were available, performed under identical conditions. Both were combined to create this test. The anomalies in these datasets were challenging to identify, and experts sometimes disagreed during labeling. Therefore, this experiment was considered a *hard test*, as any region identified as anomalous by at least one expert was labeled as such.

The first acquisition contained 3,430 records with 9 identified anomalies, while the second contained 3,764 records, also with 9 anomalies.

The anoamlies in the first acquisition has been located by domain experts as follows:

- Anomaly n°1: immediately after the station *Marche*;
- Anomaly n°2: between the stations *Massaua* and *Pozzo Strada*;
- Anomaly n°3: between the stations *Pozzo Strada* and *Monte Grappa*;
- Anomaly n°4: immediately after the station *Monte Grappa*;
- Anomaly n°5: immediately after the station *Rivoli*;
- Anomaly n°6: between the stations *Bernini* and *Principi d'Acaja*;
- Anomaly n°7: immediately after *Principi d'Acaja*;
- Anomaly n°8: between stations *Re Umberto* and *Porta Nuova*;
- Anomaly n°9: near the station *Carducci*;

While, in the second acquisition, the domain experts identified the following anomalies:

- Anomaly n°1: immediately after *Paradiso* station;
- Anomaly n°2: near the station *Marche*;

- Anomaly n°3: near the station *Monte Grappa* ;
- Anomaly n°4: immediately before *Rivoli* station;
- Anomaly n°5: immediately after *Rivoli* station;
- Anomaly n°6: immediately after *Racconigi* station;
- Anomaly n°7: between stations *Bernini* and *Principi d'Acaja*;
- Anomaly n°8: immediately before *Re Umberto* station;
- Anomaly n°9: between stations *Marconi* and *Nizza*;

Moreover, the combined dataset was divided as follows:

- Train set: composed by 3,523 records;
- Validation set: with 1,303 samples and 6 anomalies;
- Test set: composed by 2,368 records and 12 anomalies.

Test 4 Given that the magnitude of anomalies may influence the reconstruction error in certain models, this test reused the same data as Test 3 but changed the allocation of anomalies between the validation and test sets. Specifically, anomalies that were most difficult to detect, even for domain experts, were placed in the validation set to provide a more realistic threshold-setting scenario. The resulting dataset was composed as follows:

- Train set: composed by 3,523 records, the same train set of the previous test;
- Validation set: with 1,747 samples and 7 anomalies;
- Test set: composed by 1,924 records and 11 anomalies.

Test 5 This test also used the BLUE backbone data but focused only on anomalies where both experts agreed, making it a comparatively *simpler test*. Anomalies with disagreement were removed, resulting in 10 total anomalies (8 from the first acquisition and 2 from the second). The final dataset was as follows:

- Train set: composed by 2,688 records;
- Validation set: with 1,955 samples and 5 anomalies;
- Test set: composed by 2,551 records and 5 anomalies.

Test 6 In this test, data from the multimedia (GREEN) backbone were used. The acquisition consisted of 30,654 records collected during a trip from *Marche* to *Italia 61*. The final portion of the trip, when the train remained stationary at the last station, was included in the training data, as the corresponding steady-state signals represented normal behavior. Experts identified 15 anomalies:

- Anomaly n°1: between *Rivoli* and *Racconigi*;
- Anomaly n°2: between *Rivoli* and *Racconigi*;
- Anomaly n°3: between *Rivoli* and *Racconigi*;
- Anomaly n°4: between stations *Racconigi* and *Bernini*;
- Anomaly n°5: between stations *Racconigi* and *Bernini*;
- Anomaly n°6: immediately before *Bernini* station;
- Anomaly n°7: immediately after *Bernini* station;
- Anomaly n°8: between stations *Bernini* and *Principi d'Acaja*;
- Anomaly n°9: between stations *Vinzaglio* and *Re Umberto*;
- Anomaly n°10: between stations *Porta Nuova* and *Marconi*;

- Anomaly n°11: between stations *Nizza* and *Dante*;
- Anomaly n°12: immediately after *Dante* station;
- Anomaly n°13: immediately after *Dante* station;
- Anomaly n°14: between stations *Dante* and *Carducci-Molinette*;
- Anomaly n°15: immediately after *Lingotto* station.

Given these anomalies we built the dataset as usual, considering a train set of not anomalous samples and dividing anomalies between validation and test, at the end we had the following dataset:

- Train set: composed by 18,151 records, comprising also the final part of the acquisition with the steady signals;
- Validation set: with 2,898 samples and 5 anomalies;
- Test set: composed by 9,605 records and 10 anomalies.

Test 7 This acquisition differed from the others as it used data from the OBM installed at the *tail* of the train instead of the head, again from the GREEN backbone. A total of 28,377 records were collected during a trip from *Lingotto* to *Fermi*, with 39 anomalies identified:

- Anomalies from n°1 to n°3: between stations *Carducci-Molinette* and *Dante*;
- Anomalies n°4 and n°5: immediately after *Dante* station;
- Anomaly n°6: between stations *Nizza* and *Marconi*;
- Anomalies from n°7 to n°9: between stations *Marconi* and *Porta Nuova*;
- Anomalies from n°10 to n°17: between stations *Porta Nuova* and *Re Umberto*;

- Anomaly n°18: immediately after *Vinzaglio* station;
- Anomaly n°19: between stations *Vinzaglio* and *Porta Susa*;
- Anomalies n°20 and n°21: immediately before *Porta Susa* station;
- Anomaly n°22: between stations *Porta Susa* and *XVIII Dicembre*;
- Anomaly n°23: immediately before *XVIII Dicembre* station;
- Anomalies from n°24 to n°26: between stations *XVIII Dicembre* and *Principi d'Acaja*;
- Anomaly n°27: immediately before *Principi d'Acaja* station;
- Anomalies n°28 and n°29: immediately after *Principi d'Acaja* station;
- Anomalies n°30 and n°31: between stations *Racconigi* and *Rivoli*;
- Anomaly n°32: between stations *Rivoli* and *Monte Grappa*;
- Anomaly n°33: immediately before *Pozzo Strada* station;
- Anomalies from n°34 to n°36: immediately before *Marche* station;
- Anomaly n°37: between stations *Marche* and *Paradiso*;
- Anomaly n°38: immediately before *Paradiso* station;
- Anomaly n°39: between stations *Paradiso* and *Fermi*;

The dataset composition was as follows:

- Train set: composed by 21,150 records;
- Validation set: with 715 samples and 6 anomalies;
- Test set: composed by 6,512 records and 33 anomalies.

Test 8 In this test, we again employed data from the multimedia (GREEN) backbone. Specifically, the telecommunication engineers collected data from a limited portion of the railway line, using a high sampling frequency to ensure a fine temporal resolution. The acquisition covered the train journey from *Bernini* station to *Marche* station, resulting in a total of 25,192 records gathered by the OBM installed at the head of the train.

This acquisition is particularly noteworthy because, at the beginning of the data collection, the on-board OBM initially connected to a radio access point located between the stations of *Racconigi* and *Rivoli*. Subsequently, it connected to access points near stations preceding *Racconigi*, and later to one situated between *Principi d'Acaja* and *Bernini*. This unusual sequence suggests that during communication between the backbone and the OBM some issues occured.

During the expert labelling phase, the domain specialists identified a total of 13 anomalies occurring throughout the trip. The anomalies are listed below in chronological order, taking into account the atypical hand-off sequence:

- Anomaly n°1: between stations *Rivoli* and *Racconigi*;
- Anomaly n°2: between stations *Racconigi* and *Bernini*;
- Anomalies n°3 to n°6: immediately after *Bernini* station;
- Anomalies n°7 to n°9: between stations *Racconigi* and *Rivoli*;
- Anomaly n°10: the OBM on board connected to four different access points in less than one minute, causing multiple transient connections with TREBOX along various positions of the railway;
- Anomalies n°11 to n°13: immediately before *Marche* station.

This acquisition exhibited particularly irregular behaviour, especially the last anomalies (n°10 to n°13), which were caused by communication issues between the on-board OBM and the backbone during the journey. Consequently, this test was valuable to assess whether the anomaly detection methods could correctly identify such atypical anomalies.

As usual, we constructed the dataset based on this acquisition, obtaining the following data splits:

- Train set: 16,729 records;
- Validation set: 2,814 samples including 5 anomalies;
- Test set: 5,649 records including 8 anomalies.

Test 9 In this final test, we aimed to investigate whether it was possible to train the anomaly detection methods exclusively on synthetic data, in order to assess and potentially enhance their capability in identifying anomalies. Specifically, we employed the same data used in Test 3, corresponding to the BLUE backbone. In this case, we maintained the same labelling strategy adopted in Test 3, considering as anomalous all regions in the radio signals identified by at least one of the domain experts.

To this end, we employed our synthetic data generation pipeline to produce a complete training set consisting solely of non-anomalous synthetic data. The validation and test sets remained identical to those used in Test 3. In particular, the training set comprised a total of 30,000 synthetic records.

5.2.5 Feature Selection

In the field of Anomaly Detection, it is often crucial to implement an appropriate *Feature Selection* process. Indeed, some features may be redundant or introduce noise, ultimately degrading the performance of anomaly detection models. A vast number of feature selection techniques exist in the literature, each with its own advantages and limitations. In this work, several distinct feature selection methods were employed and compared to determine whether any of them could yield better detection performance across different models.

To conduct this analysis, all experiments described in subsection 5.2.4 were repeated, but this time including only the features selected by each feature selection method. Hence, for each feature subset, the same anomaly detection tests were executed. The algorithms and evaluation metrics used were identical to those presented in the aforementioned section, ensuring consistency in the comparative analysis.

Feature selection was also performed separately for each dataset, depending on the data source; namely the RED, BLUE, or GREEN backbone. As a consequence, the selected features were not necessarily the same across all datasets: in some cases, the methods identified distinct subsets of features for different backbones. The selected features for each backbone and selection method are summarized in Appendix B.

Domain Expert Technique During the generation and validation of synthetic data, particularly while assessing the quality of the generated samples, it was observed that the Random Forest model assigned significantly higher importance to four specific features: *Throughput (Mbps)*, *One-Way Delay (ms)*, *Jitter (delay variation) Maximum (ms)*, and *RSSI*. These features were also independently identified by domain experts as the most informative variables, representing those most frequently analyzed during the data labeling process.

For this reason, an additional test was conducted using only these four features, with the goal of evaluating whether it was possible to substantially reduce the dimensionality of the dataset while preserving, or even improving, the model's detection performance. This setup thus serves as a benchmark for assessing the trade-off between model simplicity and detection capability, leveraging both data-driven insights and expert knowledge.

Principal Component Analysis (PCA) Another widely adopted technique in the literature for feature selection is the *Principal Component Analysis (PCA)*. PCA is an unsupervised statistical method that projects the original features into a new set of orthogonal axes, called *principal components*, which are ordered according to the amount of variance they explain in the data. The first principal component captures the largest possible variance, while each subsequent component accounts for the maximum remaining variance under the constraint of being orthogonal to all previous components.

In the proposed approach, the PCA model is fitted to the standardized data without predefining the number of components, allowing the method to compute the full spectrum of eigenvectors and eigenvalues of the covariance matrix. The cumulative explained variance is then analyzed to determine the smallest number of components required to retain a specified proportion of the total variance; in this case, 95%.

Once the relevant components have been identified, their *loadings*, which quantify the contribution of each original feature to the corresponding principal components, are computed. Features with the highest absolute loadings across the retained components are considered the most informative, as they contribute the most to explaining the variability captured by the PCA model. These features are therefore selected to form a reduced and representative subset of the original dataset.

Correlation-Based Feature Selection (CFS) Another unsupervised approach employed for dimensionality reduction is the *Correlation-Based Feature Selection (CFS)* method. This technique aims to remove redundant variables by analyzing the pairwise correlations among features and discarding those that exhibit a high degree of linear dependency. The underlying rationale is that highly correlated features tend to convey overlapping information, which can inflate model complexity without improving predictive performance.

In the proposed implementation, the absolute correlation matrix of the dataset is computed. The collection of all unique correlation coefficients is then sorted in ascending order and examined through the *elbow method* to automatically determine an optimal correlation threshold. The elbow point, identified using the `KneeLocator` algorithm, represents the value beyond which correlations increase sharply, indicating excessive redundancy among features. If the elbow point cannot be clearly identified, a fallback strategy is applied by selecting the 90th percentile of the correlation values as the threshold.

Once the optimal threshold has been established, all features whose correlations exceed this limit are considered redundant and are consequently removed from the dataset.

K-Means Clustering-Based Feature Selection The last unsupervised approach adopted in this study for dimensionality reduction is based on *K-Means clustering*. Unlike correlation, or variance-based methods, this technique leverages the clustering structure of the data to assess the relative importance of each feature. The underlying assumption is that features that better preserve the separation between clusters are more informative and discriminative, whereas features whose distributions do not vary significantly across clusters contribute little to the overall data structure.

The *K-Means* algorithm is applied to the dataset to partition the samples into k clusters, where k is a predefined number of centroids. In our case study we set k equal to 3, the idea is that the data can be divided into three different sets: one containing the anomalous records, a second with the not anomalous samples and a last group consisting in those records that are between the first two. For each feature, the mean absolute distance between its values within each cluster and the corresponding centroid is computed. This distance represents how much a given feature contributes to distinguishing the data points within clusters. A feature whose values are close to their cluster centroid exhibits low variability and therefore provides limited discriminative power, whereas features showing larger deviations indicate stronger separation capabilities and are ranked higher.

Each feature is thus assigned an importance score equal to its average distance from all cluster centroids. The features are ranked in descending order according to this score, forming a *feature importance curve*. To determine how many top features (k_{top}) should be retained, the *elbow method* is applied to this curve using the *KneeLocator* algorithm. The elbow point represents the stage beyond which adding further features yields diminishing returns in terms of added information. If no clear elbow is detected, a fallback heuristic is employed by selecting approximately one-third of the features.

Chapter 6

Experimental Results

This chapter presents and discusses the experimental results obtained from the analyses described in the previous chapter. Each section focuses on a specific aspect of the study, aiming to assess both the effectiveness of the proposed methodology and the impact of different design choices on the overall anomaly detection performance.

The first part of the chapter (section 6.1) evaluates the quality of the synthetic data generated by the selected Synthetic Data Generation (SGD) methods. Subsequently, section 6.2 presents the results obtained by the contamination factor estimation methods for Isolation Forest.

The following sections investigate two complementary aspects: the effectiveness of various thresholding strategies (section 6.3) and the influence of the ratio between real and synthetic data (section 6.4) on the performance of the detection models. After these analyses, section 6.5 reports the final evaluation of the anomaly detection methods across different datasets, considering both real and hybrid (real + synthetic) data configurations.

Finally, section 6.6 presents the results related to the feature selection study, which aims to identify the most informative variables contributing to the detection of anomalous samples.

6.1 Synthetic Data Quality

During this work, several Synthetic Data Generation (SDG) methods were tested with the aim of identifying the best-performing one. This section presents the results of the conducted evaluation, which is described in detail in section 5.1.

It is important to note that some generation models can be trained to respect the inherent constraints in the data. For those models that are not explicitly designed for this purpose, an SMT solver was employed to enforce such constraints on the generated data. An exception is SMOTE: due to its nature, applying the SMT solver to its output is meaningless, since this method inherently tends to reproduce real data directly.

Each SDG method was evaluated using distance-based metrics between real and synthetic samples. Additionally, the generated data were visually inspected by comparing feature distributions and analyzing latent-space representations obtained via UMAP and t-SNE. Finally, a Discriminator Two-Sample Test (D2ST) was performed by training a Random Forest classifier to distinguish between real and synthetic samples.

Distance Metrics and D2ST Table 6.1 summarizes the results obtained from the evaluation of the quality of synthetic data generated by the different models. In terms of MMD, the best-performing models are ForestDiffusion and DoppleGANger (both with 0.004), followed by TVAE (0.006) and SMOTE (0.010). These results indicate that these models are capable of closely reproducing the statistical distribution of the real data. Conversely, TimesFM (0.382) and TimesFM + SMT (0.150) show significantly higher discrepancies, suggesting a weaker ability to generalize the underlying data distribution.

Regarding the EMD metric, the lowest value is achieved by SMOTE (0.953), although this should be interpreted cautiously, since SMOTE tends to replicate samples directly from the real data, thus reducing diversity. Models such as ForestDiffusion (1.442) and GMM (1.611) also show good similarity, while

Generation Method	**MMD ↓**	**EMD ↓**	**DTW ↓**	**D2ST (Acc.) ↓**
SMOTE	0.010	$\mathbf{0.953}$	$\mathbf{5 \times 10^3}$	86%
DTWBA	0.080	5.014	9×10^3	98%
DTWBA + SMT	0.107	7.075	1.6×10^4	100%
GMM	0.017	1.611	2.1×10^4	94%
GMM + SMT	0.039	6.842	1.7×10^4	100%
ForestDiffusion	$\mathbf{0.004}$	1.442	7×10^3	84%
DoppleGANger	$\mathbf{0.004}$	5.757	1.7×10^4	99%
DoppleGANger + SMT	0.051	7.859	1.6×10^4	100%
CTGAN	0.016	5.767	1.7×10^4	86%
TVAE	0.006	5.378	1.7×10^4	$\mathbf{66}\%$
PAR	0.067	9.463	1.3×10^4	98%
PAR + SMT	0.067	9.851	1.6×10^4	100%
Chronos	0.083	7.386	9×10^3	100%
Chronos + SMT	0.094	7.298	1.7×10^4	99%
TimesFM	0.382	7.488	1.6×10^4	100%
TimesFM + SMT	0.150	8.561	1.6×10^4	100%

Table 6.1: Synthetic data quality evaluation for different generation methods. The table reports distance-based metrics (MMD, EMD, DTW) and the result of the Discriminator Two-Sample Test (D2ST) in terms of accuracy. Lower values indicate higher similarity to real data.

approaches like PAR, DTWBA, and Chronos report larger distances, up to 9.851 for PAR + SMT.

The DTW metric, which captures temporal discrepancies, shows consistent behavior across models. The lowest values are obtained by SMOTE (5×10^3) and DTWBA (9×10^3), while most generative models (e.g., GAN- and VAE-based) show distances in the range of 10^4–2×10^4, suggesting higher temporal variability between real and synthetic data. This behavior is expected because these models are trained to generate high-fidelity data, without taking into acount the temporal relations between samples.

The D2ST results are consistent with these findings. A lower accuracy indicates that the discriminator struggles to differentiate between real and synthetic data, implying higher quality generation. In this regard, TVAE (66%) and ForestDiffusion (84%) emerge as the best-performing models. On the contrary, models such as DTWBA + SMT, GMM + SMT, and TimesFM + SMT reach 100% accuracy, showing that their synthetic samples are easily distinguishable from real ones.

The inclusion of the SMT (Satisfiability Modulo Theories) constraint does not consistently improve data quality. In several cases (e.g., DTWBA + SMT, GMM + SMT, DoppleGANger + SMT), the addition of SMT solver to change the samples, increases the distances and D2ST accuracy.

Overall, the results indicate that:

- ForestDiffusion and TVAE provide the best trade-off between statistical similarity and realistic data generation.
- SMOTE produces highly similar samples but lacks diversity.
- The addition of the SMT constraint ensures logical consistency but often degrades statistical fidelity.
- Despite their complexity, transformer-based models such as TimesFM and Chronos do not outperform simpler approaches in reproducing the true data distribution.

Latent Space Analysis Latent space analysis provides an intuitive visualization of how closely the synthetic data distributions align with real data in a reduced-dimensional representation. Figure 6.1 shows the latent space representations of real and synthetic data generated by each model. Synthetic samples are depicted in orange, real samples in blue, and anomalous real data in red, included for comparison. For each model, both *t-SNE* (left) and *UMAP* (right) projections are reported.

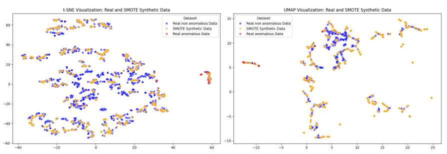
From the latent space visualizations, it is evident that the SMOTE model tends to generate samples that are almost replicas of real data, thereby reducing statistical variability. This behavior is visible from the near-complete overlap between real and synthetic points. Furthermore, for all models, the introduction of the SMT solver generally leads to the generation of data that do not share the same latent space with real samples, as clearly observed in the cases of DTWBA and GMM.

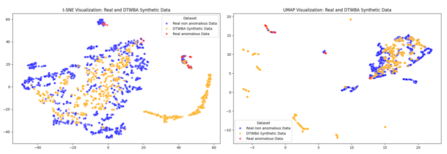
Among the tested models, those that produce synthetic data sharing a common latent representation with real samples are GMM, ForestDiffusion, CTGAN, and TVAE, while SMOTE and DTWBA show only partial overlap. For all remaining models, the synthetic data occupy a distinctly different latent space compared to the real one.

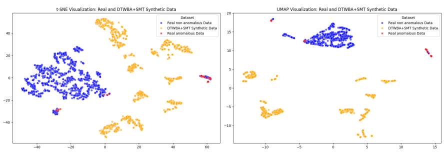
Interestingly, the generative models CTGAN and TVAE tend to produce synthetic data that are well separated from the anomalous real samples in the latent space. This suggests a potentially higher quality and usefulness of these models for the specific anomaly detection use case of this study. Such separation, however, is not observed for SMOTE, DTWBA, GMM, or ForestDiffusion.

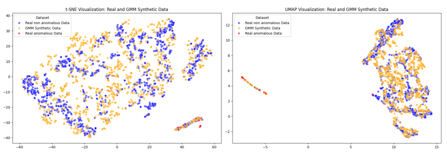
Overall, models that exhibit greater latent overlap with real data are expected to achieve better downstream performance in anomaly detection tasks.

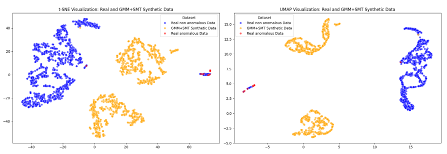
Distribution Analysis As previously introduced, for each SDG method we also performed a visual inspection of the feature distributions, comparing real

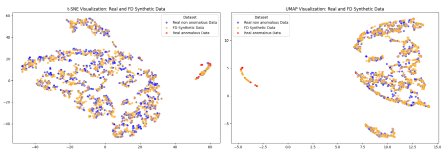

(a) SMOTE

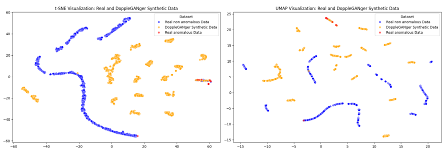

(b) DTWBA

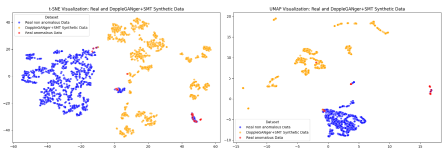

(c) DTWBA + SMT

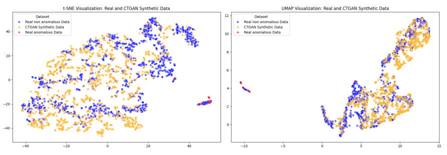

(d) GMM

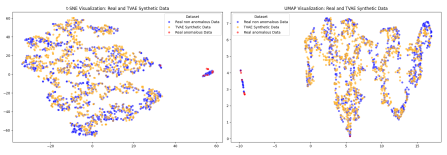

(e) GMM + SMT

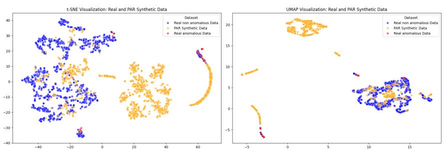

(f) ForestDiffusion

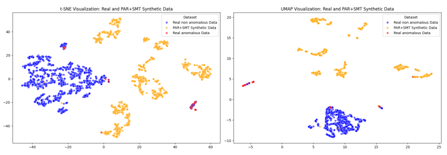

(g) DoppleGANger

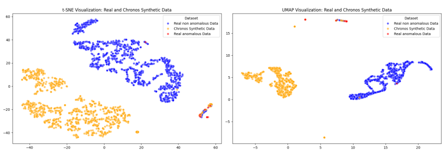

(h) DoppleGANger + SMT

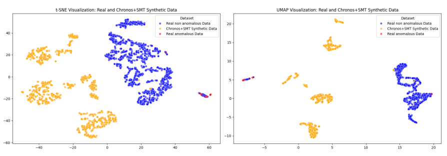

(i) CTGAN

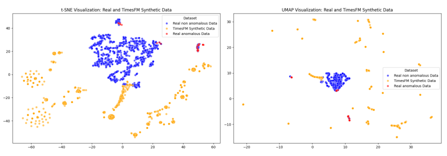

(j) TVAE


(k) PAR


(l) PAR + SMT


(m) Chronos


(n) Chronos + SMT


(o) TimesFM

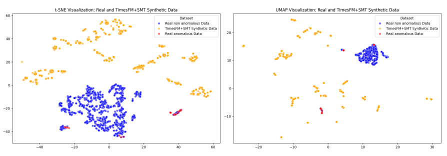
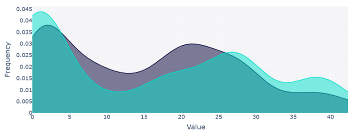
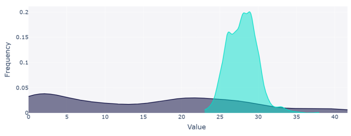

(p) TimesFM + SMT

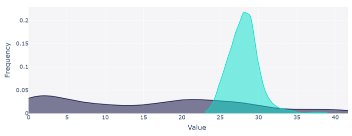
Figure 6.1: Visualization of the latent spaces generated by different synthetic data generation methods using t-SNE and UMAP. The plots depict synthetic data in orange, real data in blue, and anomalous samples in red, included for comparison purposes.



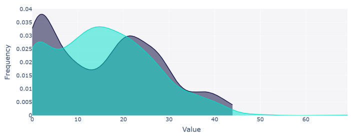
(a) SMOTE



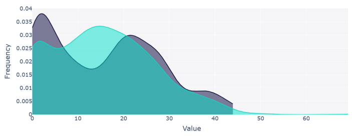
(b) DTWBA



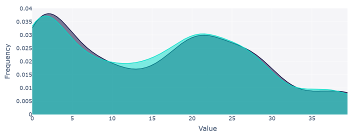
(c) DTWBA + SMT



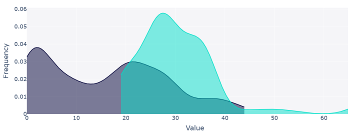
(d) GMM



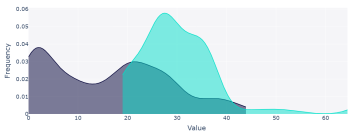
(e) GMM + SMT



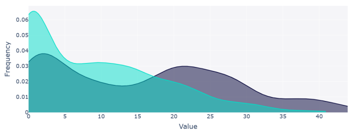
(f) ForestDiffusion



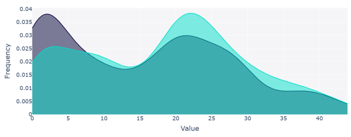
(g) DoppleGANger



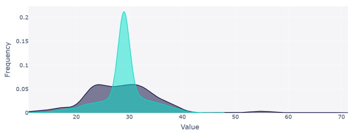
(h) DoppleGANger + SMT



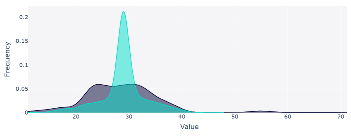
(i) CTGAN



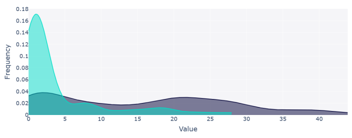
(j) TVAE



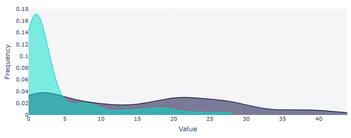
(k) PAR



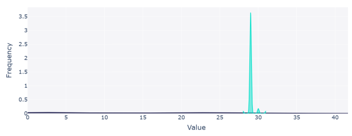
(l) PAR + SMT



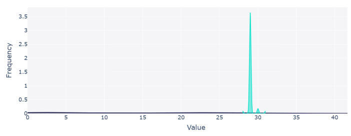
(m) Chronos



(n) Chronos + SMT



(o) TimesFM



(p) TimesFM + SMT

Figure 6.2: Visualization of *RSSI* signal distribution for both real and synthetic data generated by several SGD methods. In light blue the synthetic data distribuzion, in gray the real one. Notably, the scale is normalized.

and synthetic data. Figure 6.2 shows an example of these distributions for the *RSSI* signal across different generation methods.

From the visual analysis, it emerged that the best overlap between real and synthetic data was obtained with the SMOTE, ForestDiffusion, and TVAE models. Conversely, DTWBA, PAR, Chronos, and TimesFM struggled to reproduce distributions that closely match the real data. Similar trends were observed for the other features of the dataset.

Interestingly, the inclusion of the SMT solver did not significantly affect the overall distribution shapes, indicating that its main impact lies in enforcing logical constraints rather than improving statistical fidelity.

Overall Discussion Across all evaluation perspectives, distance metrics, latent space visualization, and feature distribution analysis, consistent trends emerge. The ForestDiffusion and TVAE models stand out as the most balanced approaches, generating synthetic data that closely resemble real samples both statistically and visually, while maintaining sufficient variability. SMOTE also achieves low distance metrics, but its tendency to replicate real samples limits its usefulness for generating truly novel data.

The introduction of the SMT solver, although valuable for enforcing logical consistency, generally does not improve the statistical similarity of the generated data and sometimes even degrades it.

Overall, models such as TVAE, ForestDiffusion, and to some extent CTGAN emerge as the most promising candidates for generating synthetic data for downstream tasks like anomaly detection. In particular, TVAE seems to be able to fool the Random Forest classifier during the D2ST more respect to other models.

6.2 Contamination Factor Estimation

As previously discussed, one of the main challenges of the *Isolation Forest* algorithm lies in identifying a robust strategy for selecting the *contamination factor* hyperparameter, which determines the expected proportion of anomalies in the dataset. To address this, several estimation methods were tested on real data. Five independent experiments were performed, each including data from all different backbones (RED, BLUE and GREEN) to ensure a fair and comprehensive comparison. The design of these tests and the evaluation metrics used are described in subsection 5.2.1 of chapter 5.

Table 6.2 reports the results obtained for each test. The tested methods are: *Automatic Estimation (AE)*, *GammaGMM (γGMM)*, *Domain-based Heuristic on Samples (DBH-S)*, *Domain-based Heuristic on TREBOXES (DBH-T)* and *Zero Anomalies Assumption (ZAA)*.

Checking the results, we can note that ZAA consistently shows the best overall performance across almost all tests. In *Test 1*, it achieves the highest Accuracy (0.98), F1-score (0.91), and MCC (0.90), indicating both excellent precision and balance between true and false predictions. Similarly, in *Tests 2, 4, and 5*, ZAA maintains leading or tied-best values in key metrics such as Accuracy, F1, and MCC, confirming its robustness and stability across different settings.

The AE and γGMM methods exhibit similar intermediate performance, particularly in their ability to perfectly identify positive cases (Recall = 1.00 across all tests) and maintain a low False Negative Rate (FNR = 0). However, both models tend to produce a higher number of false positives, which reduces their overall Precision and MCC. Notably, γGMM slightly outperforms AE in several instances (e.g., *Test 3* and *Test 5*) due to a better balance between precision and recall, leading to higher F1 and MCC values.

The DBH-S and DBH-T variants perform more variably. While both achieve perfect True Negative detection (high TN) in several tests (e.g., *Test 1* and *Test*

Test	Method	TP	FN	TN	FP	FNR	Acc.	Prec.	Rec.	F1	MCC
	AE	**5**	**0**	12	29	**0**	0.37	0.15	**1.00**	0.26	0.21
	γGMM	**5**	**0**	26	15	0	0.67	0.25	**1.00**	0.40	0.40
Test 1	DBH-S	2	3	**41**	**0**	0.60	0.93	**1.00**	0.40	0.57	0.61
	DBH-T	**5**	**0**	6	35	0	0.24	0.12	**1.00**	0.22	0.14
	ZAA	**5**	**0**	40	1	0	**0.98**	0.83	**1.00**	**0.91**	**0.90**
	AE	**8**	**0**	1	5	**0**	0.64	0.62	**1.00**	0.76	0.32
	γGMM	**8**	**0**	1	5	**0**	0.64	0.62	**1.00**	0.76	0.32
Test 2	DBH-S	0	8	**6**	**0**	1.00	0.43	0	0	0	0
	DBH-T	0	8	**6**	**0**	1.00	0.43	0	0	0	0
	ZAA	7	1	6	0	0.12	**0.93**	**1.00**	0.88	**0.93**	**0.87**
	AE	**15**	**0**	12	29	**0**	0.48	0.34	**1.00**	0.51	0.32
	γGMM	13	2	19	22	0.13	0.57	0.37	0.87	**0.52**	0.30
Test 3	DBH-S	2	13	**41**	**0**	0.87	0.77	**1.00**	0.13	0.24	0.32
	DBH-T	**15**	**0**	11	30	**0**	0.46	0.33	**1.00**	0.50	0.30
	ZAA	4	11	**41**	**0**	0.73	**0.80**	1.00	0.27	0.42	**0.46**
	AE	**10**	**0**	4	29	**0**	0.33	0.26	**1.00**	0.41	0.18
	γGMM	**10**	**0**	5	28	**0**	0.35	0.26	**1.00**	0.42	0.20
Test 4	DBH-S	2	8	**33**	**0**	0.80	0.81	**1.00**	0.20	0.33	0.40
	DBH-T	**10**	**0**	1	32	**0**	0.26	0.24	**1.00**	0.38	0.08
	ZAA	6	4	**33**	**0**	0.40	**0.91**	1.00	0.60	**0.75**	**0.73**
	AE	**12**	**0**	0	9	**0**	0.57	0.57	**1.00**	0.73	0.00
	γGMM	**12**	**0**	2	7	**0**	0.67	0.63	**1.00**	0.77	0.37
Test 5	DBH-S	3	9	**9**	**0**	0.75	0.57	**1.00**	0.25	0.40	0.35
	DBH-T	**12**	**0**	0	9	**0**	0.57	0.57	**1.00**	0.73	0.00
	ZAA	8	4	**9**	**0**	0.33	**0.81**	1.00	0.67	**0.80**	**0.68**

Table 6.2: Performance of different contamination factor estimation methods across five independent tests. Each test includes data from multiple backbone configurations. Bold values indicate the best (or tied best) results for each metric.

3), their low True Positive Rate and high FNR significantly represent a limit. DBH-S, in particular, demonstrates excellent ability to avoid false positives (FP = 0) but struggles in identifying positive samples, resulting in lower F1-scores. These methods thus appear more conservative, favoring precision over recall.

Overall, ZAA emerges as the most reliable and balanced method, combining high recall with excellent precision and achieving top MCC values across nearly all tests. γGMM and AE form the second tier of strong performers, maintaining perfect recall but suffering from lower precision. DBH-S and DBH-T, despite strong TN performance, show limited effectiveness in positive case detection and yield the lowest overall metrics.

In conclusion, the ZAA approach appears to be the most suitable choice for the purposes of this work. In other words, training the Isolation Forest with a contamination factor fixed to zero yields the most stable and accurate results.

6.3 Study on Thresholding Methods

As introduced earlier, in this work several *thresholding methods* were tested using two different reconstruction signals: the *Sum of Squared Errors (SSE)* and the *Mean Absolute Error (MAE)*, both computed by the Autoencoder on three distinct datasets, one for each backbone (RED, BLUE, and GREEN). The details of the conducted experiments are presented in subsection 5.2.2, while a full description of the thresholding approaches is given in subsection 2.2.3.

When evaluating multiple thresholding strategies, it is essential to establish a reliable baseline. For this reason, the *Area Under the ROC Curve (AUROC)* was computed on each dataset and for both reconstruction signals. The AUROC results obtained using the MAE signal are reported in Table 6.3, while those derived from the SSE signal are presented in Table 6.4. From these tables, it is evident that MAE generally provides a more stable reconstruction

Backbone	**Dataset**	**AUROC**
RED	Validation	0.978
	Test	0.995
BLUE	Validation	0.863
	Test	0.911
GREEN	Validation	0.940
	Test	0.970

Table 6.3: Comparison of AUROC values across the three backbone datasets. The AUROC values are computed on *Mean Absolute Error (MAE)* as reconstruction signal for the Autoencoder.

Backbone	**Dataset**	**AUROC**
RED	Validation	0.978
	Test	0.976
BLUE	Validation	0.864
	Test	0.914
GREEN	Validation	0.880
	Test	0.891

Table 6.4: Comparison of AUROC values across the three backbone datasets. The AUROC values are computed on *Sum of Squared Errors (SSE)* as reconstruction signal for the Autoencoder.

signal, particularly for the multimedia line GREEN. For both reconstruction signals, the BLUE dataset confirms itself as the most challenging one for the anomaly detection task, as AUROC values are consistently lower than those obtained for RED and GREEN.

Thresholding Method	**Dataset**	**FNR**	**Acc.**	**F1**	**Prec.**	**Rec.**	**MCC**
Z-Score	Val.	**0.00**	0.67	0.20	0.11	**1.00**	0.27
	Test	**0.00**	0.67	0.44	0.29	**1.00**	0.42
K-Means	Val.	1.00	0.96	0.00	0.00	0.00	0.00
	Test	0.25	**0.97**	0.86	**1.00**	0.75	0.85
DBSCAN	Val.	**0.00**	0.04	0.08	0.04	**1.00**	0.00
	Test	**0.00**	0.13	0.24	0.13	**1.00**	0.00
DBS	Val.	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Test	0.25	**0.97**	0.86	**1.00**	0.75	0.85
SDIP	Val.	**0.00**	0.96	0.67	0.50	**1.00**	0.69
	Test	**0.00**	**0.97**	**0.89**	0.80	**1.00**	**0.88**

Table 6.5: Comparison of different *thresholding methods* across validation (Val) and test datasets, built on data from the RED backbone. All thresholding strategies were applied to the reconstruction signal computed as the *Mean Absolute Error (MAE)* of the Autoencoder. Each method is evaluated using the *False Negative Rate (FNR)*, *Accuracy*, *F1 Score*, *Precision*, *Recall*, and the *Matthews Correlation Coefficient (MCC)*. Bold values indicate the best results for each metric.

Regarding the evaluation of thresholding methods on the RED backbone using MAE as the reconstruction signal, Table 6.5 clearly shows that the best-performing method is the *Standard Deviation Inflection Point (SDIP)*, which performs nearly perfectly, achieving an FNR equal to zero and the highest F1 Score and MCC on the test set. In contrast, on the validation set, the *Difference Between Sets (DBS)* method achieves perfect classification results, making it the top performer in that stage. A particularly poor method proved to be DBSCAN, which performed poorly on both validation and test sets. Specifically, both Z-Score and DBSCAN achieve a Recall of 1.00, but DBSCAN yields a null MCC and Z-Score a low one (0.27). These two methods tend to generate a large number of false positives, likely due to an excessively low threshold set on the reconstruction error.

Thresholding Method	Dataset	FNR	Acc.	F1	Prec.	Rec.	MCC
Z-Score	Val.	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Test	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
K-Means	Val.	1.00	0.96	0.00	0.00	0.00	0.00
	Test	0.25	0.97	0.86	**1.00**	0.75	0.85
DBSCAN	Val.	**0.00**	0.04	0.08	0.04	**1.00**	0.00
	Test	**0.00**	0.13	0.24	0.13	**1.00**	0.00
DBS	Val.	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Test	0.25	0.97	0.86	1.00	0.75	0.85
SDIP	Val.	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Test	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

Table 6.6: Comparison of different *thresholding methods* across validation (Val) and test datasets, built on data from the RED backbone. All thresholding strategies were applied to the reconstruction signal computed as the *Sum of Squared Errors (SSE)* of the Autoencoder. Each method is evaluated using the *False Negative Rate (FNR)*, *Accuracy*, *F1 Score*, *Precision*, *Recall*, and the *Matthews Correlation Coefficient (MCC)*. Bold values indicate the best results for each metric.

When considering the reconstruction signal SSE for the RED backbone (Table 6.6), the overall performance profile changes significantly. Both Z-Score and SDIP perform perfectly on validation and test sets, while DBS performs almost perfectly on validation and maintains high performance on the test set. DBSCAN remains the method with the lowest scores across all metrics, while K-Means displays a similar behavior to that observed with MAE: good results on the test set but poor ones on the validation set.

The results on the RED dataset suggest that SSE is a better reconstruction signal, as it enables a clearer separation between anomalous and nominal samples. This separation allows thresholding methods to set more effective cut-off values for anomaly identification, which is less evident when using MAE.

In summary, for the RED dataset, SDIP emerges as the most stable thresholding method, achieving high performance with both reconstruction signals, while SSE appears to be the preferable choice overall.

Thresholding Method	Dataset	FNR	Acc.	F1	Prec.	Rec.	MCC
Z-Score	Val.	0.64	**0.75**	0.53	**1.00**	0.36	**0.51**
	Test	0.33	**0.81**	**0.76**	0.89	0.67	0.62
K-Means	Val.	0.82	0.68	0.31	**1.00**	0.18	0.34
	Test	0.67	0.69	0.50	**1.00**	0.33	0.46
DBSCAN	Val.	**0.00**	0.39	**0.56**	0.39	**1.00**	0.00
	Test	**0.00**	0.46	0.63	0.46	**1.00**	0.00
DBS	Val.	0.82	0.68	0.31	**1.00**	0.18	0.34
	Test	0.67	0.69	0.50	**1.00**	0.33	0.46
SDIP	Val.	0.64	**0.75**	0.53	**1.00**	0.36	**0.51**
	Test	0.42	**0.81**	0.74	**1.00**	0.58	**0.66**

Table 6.7: Comparison of different *thresholding methods* across validation (Val) and test datasets, built on data from the BLUE backbone. All thresholding strategies were applied to the reconstruction signal computed as the *Mean Absolute Error (MAE)* of the Autoencoder. Each method is evaluated using the *False Negative Rate (FNR)*, *Accuracy*, *F1 Score*, *Precision*, *Recall*, and the *Matthews Correlation Coefficient (MCC)*. Bold values indicate the best results for each metric.

As previously mentioned, the evaluation was also carried out on the dataset generated using the BLUE backbone, whose results are presented in Table 6.7. These results show that no method achieves perfect performance; however, Z-Score and SDIP once again stand out as the best-performing approaches in terms of F1 Score and MCC. A similar trend is observed for FNR, except for DBSCAN, which produces extremely poor results across all metrics. In particular, DBSCAN causes the classifier to collapse, yielding both FNR and MCC equal to zero. K-Means and DBS show comparable results, although both underperform relative to Z-Score and SDIP.

When SSE is used as the reconstruction signal on the BLUE dataset (Table 6.8), all metrics are noticeably lower. This indicates that SSE is not an optimal choice in this case, as it fails to enhance the separation between anomalies and nominal data. Here, SDIP remains the best-performing method, while Z-Score shows a slight degradation compared to MAE. Once again, DBSCAN

Thresholding Method	Dataset	FNR	Acc.	F1	Prec.	Rec.	MCC
Z-Score	Val.	0.82	0.68	0.31	**1.00**	0.18	0.34
	Test	0.53	0.73	0.59	**1.00**	0.42	0.53
K-Means	Val.	0.91	0.64	0.17	**1.00**	0.09	0.24
	Test	0.75	0.65	0.40	**1.00**	0.25	0.39
DBSCAN	Val.	1.00	0.62	0.00	0.00	0.00	0.00
	Test	0.92	0.58	0.15	**1.00**	0.08	0.22
DBS	Val.	0.91	0.64	0.17	**1.00**	0.09	0.24
	Test	0.83	0.62	0.29	**1.00**	0.17	0.31
SDIP	Val.	**0.64**	**0.75**	**0.53**	**1.00**	**0.36**	**0.51**
	Test	**0.50**	**0.77**	**0.67**	**1.00**	**0.50**	**0.59**

Table 6.8: Comparison of different *thresholding methods* across validation (Val) and test datasets, built on data from the BLUE backbone. All thresholding strategies were applied to the reconstruction signal computed as the *Sum of Squared Errors (SSE)* of the Autoencoder. Each method is evaluated using the *False Negative Rate (FNR)*, *Accuracy*, *F1 Score*, *Precision*, *Recall*, and the *Matthews Correlation Coefficient (MCC)*. Bold values indicate the best results for each metric.

leads to classifier collapse, and K-Means and DBS produce similar, yet sub-optimal, outcomes.

Turning to the GREEN backbone, the results in Table 6.9 show that Z-Score and SDIP again achieve the highest overall performance on both validation and test sets. K-Means produces excellent results on the validation set but fails to generalize on the test set. Conversely, DBSCAN performs disastrously, with FNR equal to 1.00 in both splits. DBS achieves reasonably good results in terms of FNR, MCC, and F1 Score, though it still underperforms compared to SDIP and Z-Score.

When SSE is used as the reconstruction error for the GREEN backbone (Table 6.10), a slight degradation is observed across all methods on the test set. In this experiment, Z-Score demonstrates the most consistent and robust behavior across both validation and test sets, outperforming SDIP. Interestingly, DBS achieves perfect classification results in both splits, while DBSCAN once again proves unreliable, setting the threshold excessively high and leading to

Thresholding Method	**Dataset**	**FNR**	**Acc.**	**F1**	**Prec.**	**Rec.**	**MCC**
Z-Score	Val.	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Test	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
K-Means	Val.	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Test	0.50	0.94	0.67	**1.00**	0.50	0.69
DBSCAN	Val.	1.00	0.67	0.00	0.00	0.00	0.00
	Test	1.00	0.89	0.00	0.00	0.00	0.00
DBS	Val.	**0.00**	0.92	0.89	0.80	**1.00**	0.84
	Test	**0.00**	0.94	0.80	0.67	**1.00**	0.79
SDIP	Val.	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Test	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

Table 6.9: Comparison of different *thresholding methods* across validation (Val) and test datasets, built on data from the GREEN backbone. All thresholding strategies were applied to the reconstruction signal computed as the *Mean Absolute Error (MAE)* of the Autoencoder. Each method is evaluated using the *False Negative Rate (FNR)*, *Accuracy*, *F1 Score*, *Precision*, *Recall*, and the *Matthews Correlation Coefficient (MCC)*. Bold values indicate the best results for each metric.

Thresholding Method	**Dataset**	**FNR**	**Acc.**	**F1**	**Prec.**	**Rec.**	**MCC**
Z-Score	Val.	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Test	0.25	0.97	0.86	**1.00**	0.75	0.85
K-Means	Val.	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Test	0.50	0.94	0.67	1.00	0.50	0.69
DBSCAN	Val.	1.00	0.67	0.00	0.00	0.00	0.00
	Test	1.00	0.89	0.00	0.00	0.00	0.00
DBS	Val.	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Test	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
SDIP	Val.	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Test	0.50	0.94	0.67	1.00	0.50	0.69

Table 6.10: Comparison of different *thresholding methods* across validation (Val) and test datasets, built on data from the GREEN backbone. All thresholding strategies were applied to the reconstruction signal computed as the *Sum of Squared Errors (SSE)* of the Autoencoder. Each method is evaluated using the *False Negative Rate (FNR)*, *Accuracy*, *F1 Score*, *Precision*, *Recall*, and the *Matthews Correlation Coefficient (MCC)*. Bold values indicate the best results for each metric.

the maximum FNR value.

As previously discussed, we also evaluated a thresholding mechanism based on percentiles. The detailed results of this approach are reported in Appendix A. From these results, it is evident that no single percentile value consistently outperforms the other unsupervised methods across all datasets. However, overall, the percentile-based thresholding with the threshold set at the 99.9^{th} percentile achieves performance levels comparable to the Z-Score and SDIP methods on all backbones, demonstrating its effectiveness as a simple yet competitive alternative.

Conclusions Overall, the comparative analysis across all three backbones highlights several consistent trends. First, the *Standard Deviation Inflection Point (SDIP)* method consistently emerges as the most robust and reliable thresholding strategy, showing stable and often near-perfect performance across different reconstruction signals and datasets. The *Z-Score* method also demonstrates competitive results, particularly when the reconstruction error distribution is well-behaved and separable, as in the GREEN dataset.

In contrast, clustering-based approaches such as *K-Means* and *DBS* exhibit variable performance, often achieving good accuracy on the validation set but failing to generalize effectively to unseen data. *DBSCAN*, on the other hand, systematically underperforms, frequently leading to classifier collapse due to an overly aggressive threshold selection.

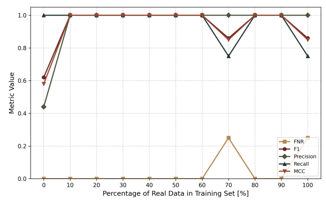
In conclusion, from the perspective of reconstruction signals, *SSE* tends to produce better separability between nominal and anomalous samples in the RED dataset, while *MAE* yields more stable behavior across all backbones, especially for GREEN.

6.4 Study on Ratio between Real and Synthetic Data

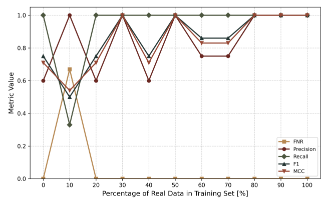
This section presents the results of the experiment concerning the variation in the ratio between real and synthetic data in the training set used to train the three anomaly detection methods employed in this work: *Isolation Forest (IF)*, *Gaussian Mixture Models (GMMs)*, and *Autoencoder (AE)*. A detailed description of the experimental setup involving data from all three backbones (RED, BLUE, and GREEN) is provided in subsection 5.2.3.

As previously discussed, each method was tested on validation and test datasets for each backbone. Regarding the results obtained for the RED backbone, shown in Figure 6.3, it can be observed that all three methods perform poorly when trained exclusively on synthetic data, that is, when no real samples are included in the training set. Analyzing the specific metric trends for each model, the Isolation Forest does not appear to exhibit a linear trend in performance as the ratio between real and synthetic data changes. Notable misclassifications occur in the validation set when the proportion of real data reaches 70% and 100%. When trained only on synthetic data, the model shows poor overall performance, except in terms of FNR. For the Gaussian Mixture Models, good performance is achieved only when the proportion of real data is high (around 80%). A slight positive trend can be observed: the higher the proportion of real data, the better the model performance. The only exception is the initial case (training on synthetic data only), which results in poor validation performance but surprisingly high performance on the test set. The Autoencoder results on the RED backbone clearly show a linear trend: as the proportion of real data increases, so do the model's performances. Specifically, excellent results on both validation and test sets are obtained when the proportion of real data reaches 80%.

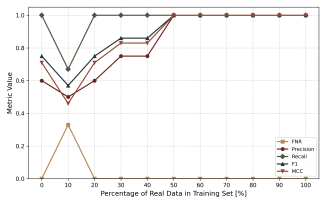
The results for the BLUE backbone are reported in Figure 6.4. It is worth recalling that this dataset contained a large number of uncertain anomalies,



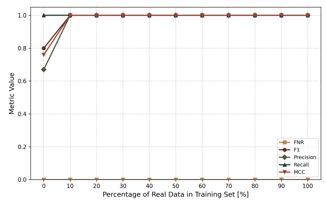
(a) IF on Val.



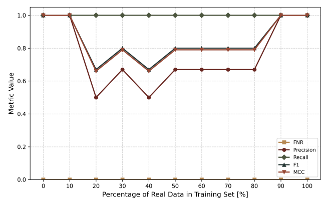
(b) GMM on Val.



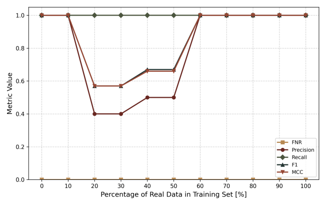
(c) AE on Val.



(d) IF on Test.



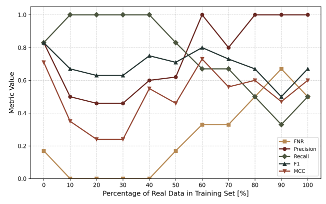
(e) GMM on Test.



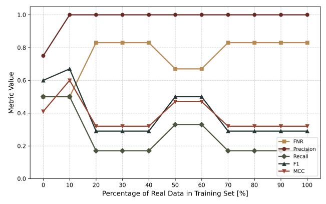
(f) AE on Test.

Figure 6.3: Performance comparison in terms of *False Negative Rate (FNR, lower is better)*, *Precision (higher is better)*, *Recall (higher is better)*, *F1 Score (higher is better)*, and *Matthews Correlation Coefficient (MCC, is better)* for the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) methods, as the ratio between real and synthetic data in the training set varies. The experiments are conducted on the RED backbone dataset.

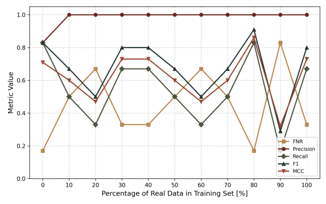
where “uncertain” refers to cases in which domain experts were often in disagreement during the labeling process. For Isolation Forest, it is interesting to note that in terms of F1 Score and MCC, the model performs best on the validation set when the percentage of real data is zero. When the ratio of real data varies between 20% and 40%, the model achieves its best results on the test set and the worst on the validation set. Moreover, when the proportion of real data exceeds 40%, the FNR shows an increasing trend on the validation set and remains consistently high (around 40%) on the test set. The GMM model performs best, on both validation and test sets, when the proportion of real data is 10%. However, as the proportion of real data increases, the performance progressively deteriorates across all metrics (FNR, F1 Score, MCC, and Recall), especially on the test set. Regarding the Autoencoder, the variation in the proportion of real data seems to have little effect on the model’s average performance on the test set. A similar pattern is observed on the validation set, although with higher oscillations. In particular, the Autoencoder performs worst when the proportion of real data reaches 90%, both on validation and test sets.



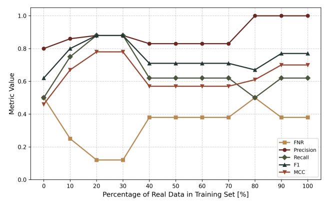
(a) IF on Val.



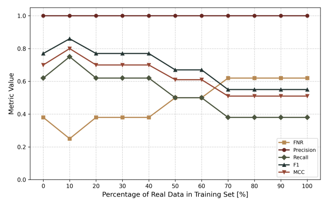
(b) GMM on Val.



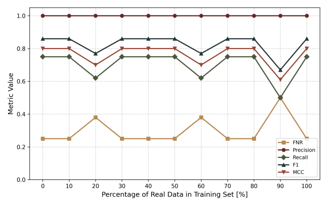
(c) AE on Val.



(d) IF on Test.



(e) GMM on Test.

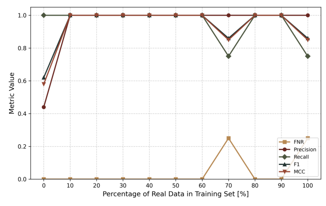


(f) AE on Test.

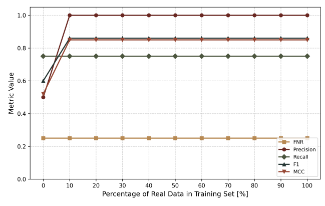
Figure 6.4: Performance comparison in terms of *False Negative Rate (FNR, lower is better)*, *Precision (higher is better)*, *Recall (higher is better)*, *F1 Score (higher is better)*, and *Matthews Correlation Coefficient (MCC, is better)* for the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) methods, as the ratio between real and synthetic data in the training set varies. The experiments are conducted on the BLUE backbone dataset.

The results obtained for the GREEN backbone are shown in Figure 6.5. In this case, it is evident that variations in the proportion of synthetic data in the training set do not significantly affect model performance. This observation holds true for both validation and test sets and across all anomaly detection methods. The only exception occurs when training is performed exclusively on synthetic data: in that case, all three models reach their lowest values in terms of F1 Score, MCC, and Precision. On the test set, both GMM and AE also exhibit the highest FNR when trained solely on synthetic data.

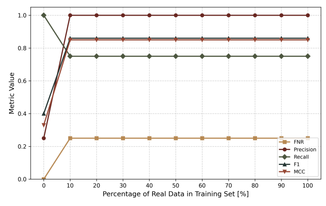
To further investigate the effects of varying the proportion of real data in the training set, we also computed the mean reconstruction error (MRE) produced by the Autoencoder. The corresponding results are shown in Figure 6.6, where the values for the validation, test, and support sets are plotted on a logarithmic scale. It is immediately apparent that when the model is



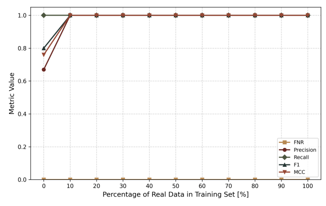
(a) IF on Val.



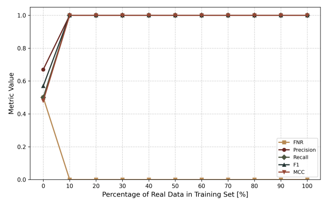
(b) GMM on Val.



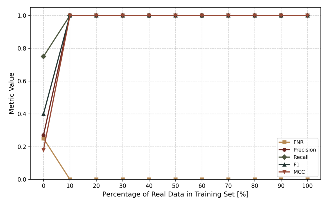
(c) AE on Val.



(d) IF on Test.



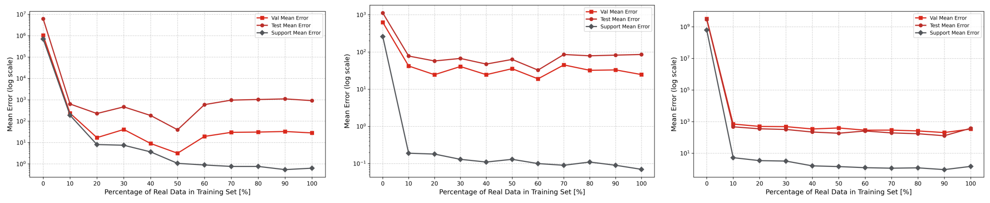
(e) GMM on Test.



(f) AE on Test.

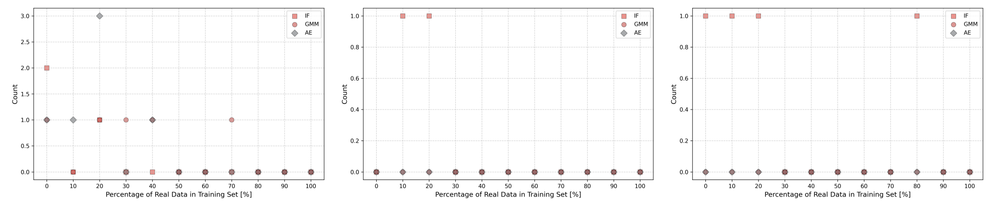
Figure 6.5: Performance comparison in terms of *False Negative Rate (FNR, lower is better)*, *Precision (higher is better)*, *Recall (higher is better)*, *F1 Score (higher is better)*, and *Matthews Correlation Coefficient (MCC, is better)* for the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) methods, as the ratio between real and synthetic data in the training set varies. The experiments are conducted on the GREEN backbone dataset.

trained exclusively on synthetic data, the mean reconstruction error is very high. For each backbone and dataset, a decreasing trend can be observed as the proportion of real data increases. Beyond 50%, the error tends to stabilize, showing minimal variation as more real data are added. In general, increasing the proportion of real data in the training set leads the Autoencoder to produce smaller reconstruction errors. After surpassing the 50% threshold, the error continues to decrease, though with diminishing returns. This suggests that the training dataset should ideally contain at least half real data. A similar trend can be observed in Figure 6.7, which reports the number of False Positives (FP) identified by each model on the support set as the ratio between real and synthetic data in the training set varies. From the figure, it can be seen that when the proportion of real data exceeds 50%, the models tend not to identify any FPs in the support set (with the exception of IF). Conversely, when the proportion of real data is low (between 0% and 20%), the models tend to identify a higher number of false positives in the support set. The highest number of FPs (three) was produced by the Autoencoder on the RED backbone.



(a) RED. **(b)** BLUE. **(c)** GREEN.

Figure 6.6: Mean Reconstruction Error (MRE) obtained by the Autoencoder as the proportion of real to synthetic data in the training set varies. The y-axis is presented in logarithmic scale. Results for validation, test, and support sets are shown for each backbone (*RED*, *BLUE*, and *GREEN*).



(a) RED. **(b)** BLUE. **(c)** GREEN.

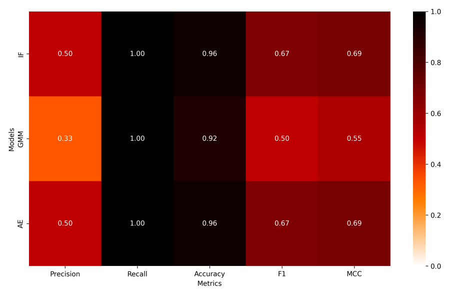
Figure 6.7: The figure shows the number of False Positives (FP) detected by each anomaly detection model, Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE), as the ratio between real and synthetic data in the training set varies. The results are reported separately for the three backbone datasets: RED, BLUE, and GREEN.

Overall, the conducted experiment highlight the critical role of real data in the training process of anomaly detection models. Across all backbones, Isolation Forest, Gaussian Mixture Models, and the Autoencoder exhibit notably degraded performance when trained exclusively on synthetic data, confirming that synthetic samples alone are insufficient to capture the complexity of real-world anomalies in our use case. The inclusion of real data consistently leads to improved results, with performance stabilizing (more or less) once the proportion of real data in the training set reaches approximately 50%. Among the three models, the Autoencoder demonstrates the most stable and interpretable behavior, showing a clear monotonic improvement in both performance metrics and reconstruction error as the ratio of real data increases. In contrast, the

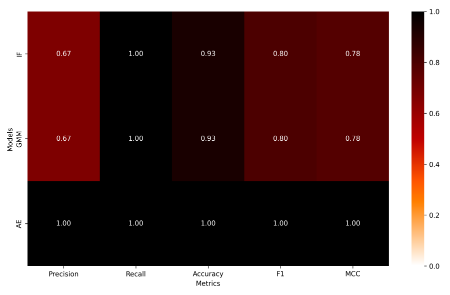
Isolation Forest and GMM models display greater variability and sensitivity to changes in the data composition. These findings suggest that, for hybrid datasets combining real and synthetic data, a balanced training composition, where real data account for at least half of the total samples, represents an effective and efficient trade-off between model generalization and data availability.

6.5 Test on Real and Synthetic Data

In this section, we present the results of the anomaly detection task conducted across a total of nine tests, each of which is described in subsection 5.2.4. The following subsections provide a detailed analysis of the results for each test.



(a) Validation.



(b) Test.

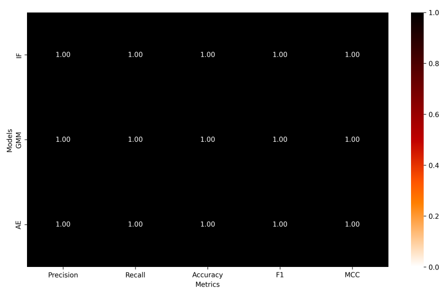
Figure 6.8: Performance of the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) in Test 1 of the anomaly detection task. For each model, the evaluation metrics include *Precision*, *Recall*, *Accuracy*, *F1-Score*, and *Matthews Correlation Coefficient (MCC)*. The results are presented through two separate heatmaps, corresponding to the validation and test datasets.

Test 1 The results of Test 1 are summarized in Figure 6.8. From these results, it is evident that all three models successfully identify every anomaly in both the *Validation* and *Test* sets. This is demonstrated by the *Recall* value of 1.0, corresponding to a *False Negative Rate (FNR)* of 0.0. Thus, all models are capable of detecting all anomalies present in both datasets.

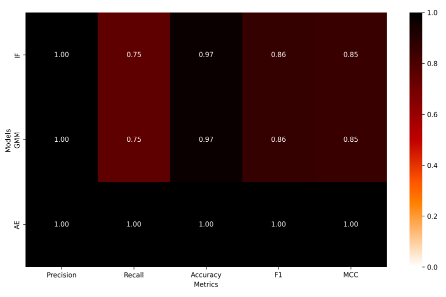
Furthermore, on the *Validation* set, the *IF* and *AE* models exhibit identical performance. In contrast, the *GMM* model produces a higher number of false positives, which results in a lower *Precision* value (0.33, compared to 0.50 for the other two models). Consequently, the *GMM* model demonstrates the weakest performance in this test across multiple metrics, including *Precision*, *F1-Score*, *Accuracy*, and *MCC*.

Regarding performance on the *Test* set, the *IF* and *GMM* models behave identically. Meanwhile, the *AE* model achieves perfect anomaly identification, obtaining a score of 1.0 for all metrics and therefore an *FNR* of 0.0.

Overall, considering both datasets, it is clear that the *AE* model outperforms the others, while the *GMM* model yields the poorest results, performing worse than *IF*.



(a) Validation.



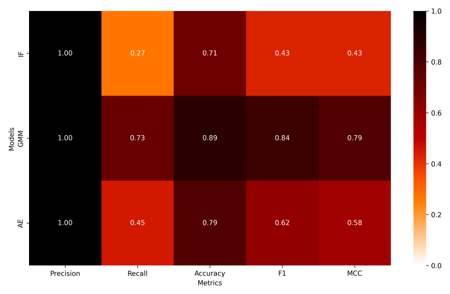
(b) Test.

Figure 6.9: Performance of the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) in Test 2 of the anomaly detection task. For each model, the evaluation metrics include *Precision*, *Recall*, *Accuracy*, *F1-Score*, and *Matthews Correlation Coefficient (MCC)*. The results are presented through two separate heatmaps, corresponding to the validation and test datasets.

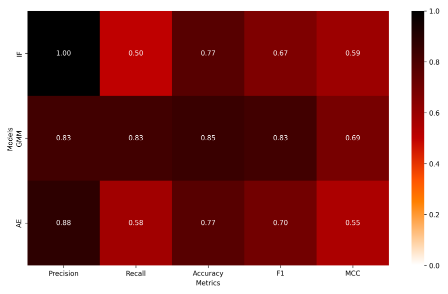
Test 2 The results for Test 2 are presented in Figure 6.9. From the heatmap corresponding to the *Validation* set, it is immediately apparent that all models achieve perfect performance, with metric values equal to 1.0 and consequently an *FNR* of 0.0.

Examining the heatmap for the *Test* set, the *AE* model once again outperforms the other two models, achieving perfect anomaly detection on this dataset as well. In contrast, the *IF* and *GMM* models fail to identify one of the anomalous APs, resulting in a *Recall* value of 0.75 and an *FNR* of 0.25.

Therefore, in this test as well, the *AE* model demonstrates superior overall performance compared to the other two models, while no clear distinction can be made between the performance of *IF* and *GMM*.



(a) Validation.



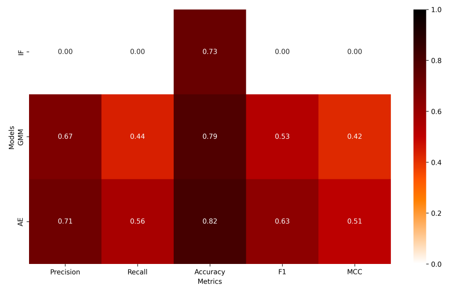
(b) Test.

Figure 6.10: Performance of the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) in Test 3 of the anomaly detection task. For each model, the evaluation metrics include *Precision*, *Recall*, *Accuracy*, *F1-Score*, and *Matthews Correlation Coefficient (MCC)*. The results are presented through two separate heatmaps, corresponding to the validation and test datasets.

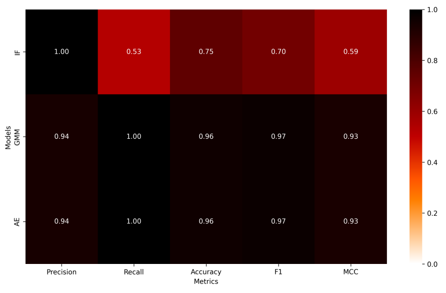
Test 3 The results obtained for Test 3 are presented in Figure 6.10. It is immediately evident that, on the *Validation* set, the *IF* model performs the worst, exhibiting the lowest *Recall* value (0.27) and therefore the highest *FNR* (0.73). Moreover, none of the models produce any false anomalies, meaning that all models correctly classify the non-anomalous APs. Interestingly, the *GMM* model achieves the lowest *FNR* on the *Validation* set (0.27), thereby outperforming both *IF* and *AE* and obtaining the highest values in terms of *Accuracy*, *F1-Score*, and *MCC*.

Regarding the *Test* set, it is clear that *IF* is the only model that does not generate any false anomalies (with a *Precision* of 1.0), but it also presents the highest *FNR* value (0.5). With respect to missed anomalies, *GMM* again appears to be the best-performing model in this set, achieving a *Recall* of 0.83, followed by *AE*, which performs significantly worse, with a *Recall* of 0.58 and thus an *FNR* of 0.42.

Overall, for this test, the *GMM* model appears to outperform the other two models in terms of *FNR*, and consequently also in *Accuracy*, *F1-Score*, and *MCC*.



(a) Validation.



(b) Test.

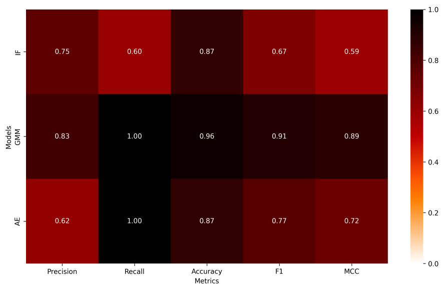
Figure 6.11: Performance of the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) in Test 4 of the anomaly detection task. For each model, the evaluation metrics include *Precision*, *Recall*, *Accuracy*, *F1-Score*, and *Matthews Correlation Coefficient (MCC)*. The results are presented through two separate heatmaps, corresponding to the validation and test datasets.

Test 4 The results of Test 4 are summarized in Figure 6.11. It can be observed that, in this case, the *IF* model performs extremely poorly, as it is unable to identify any of the anomalies present in the *Validation* set. Furthermore, the *AE* model appears to achieve the best performance on the *Validation* set across all metrics; however, none of the models succeed in performing a perfect anomaly detection. Despite being the best-performing model, *AE* still exhibits a high *FNR*, equal to 0.44 (corresponding to a *Recall* of 0.56).

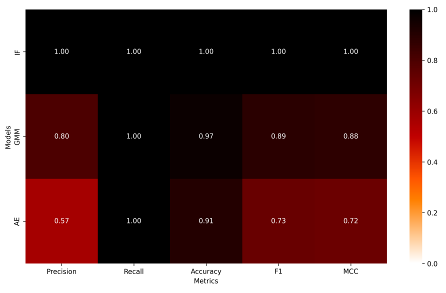
On the *Test* set, *IF* again emerges as the worst-performing model in terms of anomaly detection, presenting the lowest *Recall* value (0.53). By contrast, the *GMM* and *AE* models perform identically on this dataset: both successfully identify all anomalies (with a *Recall* of 1.0), although they exhibit a *Precision* of 0.94.

Considering the performance across both datasets, it is evident that the *AE* model performs better overall than the other two models, particularly on the

Validation set.



(a) Validation.



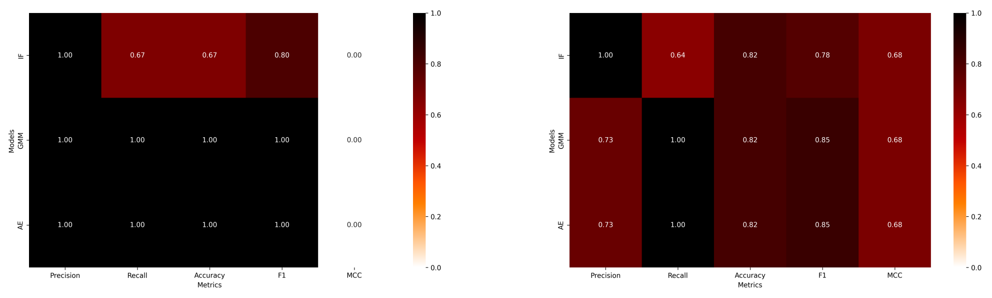
(b) Test.

Figure 6.12: Performance of the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) in Test 5 of the anomaly detection task. For each model, the evaluation metrics include *Precision*, *Recall*, *Accuracy*, *F1-Score*, and *Matthews Correlation Coefficient (MCC)*. The results are presented through two separate heatmaps, corresponding to the validation and test datasets.

Test 5 The results of Test 5 are presented in Figure 6.12. On the *Validation* set, the *IF* model obtains the worst performance in terms of *Recall*, *F1-Score*, and *MCC*, struggling to correctly identify the anomalies present in the dataset. In contrast, the *GMM* model achieves the best overall performance across all metrics.

Regarding the *Test* set, it is evident that the *IF* model performs a perfect classification, successfully identifying all anomalies. Overall, all three models correctly detect every anomaly in this dataset (achieving a *Recall* value of 1.0). However, both *GMM* and *AE* produce a number of False Anomalies; in particular, *AE* generates the highest number of such errors, resulting in the worst performance across all metrics (with the exception of *Recall*).

Considering both datasets together, *GMM* may be identified as the best-performing model, as it obtains strong *MCC* values on both the *Validation* and *Test* sets. By contrast, the *IF* model exhibits extremely poor performance on the *Validation* set while performing excellently on the *Test* set.



(a) Validation. **(b)** Test.

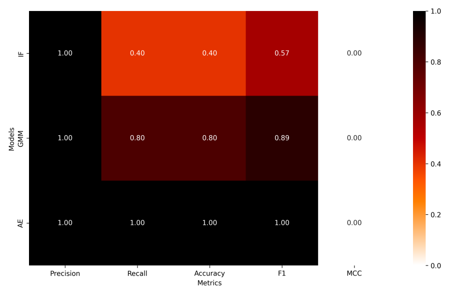
Figure 6.13: Performance of the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) in Test 6 of the anomaly detection task. For each model, the evaluation metrics include *Precision*, *Recall*, *Accuracy*, *F1-Score*, and *Matthews Correlation Coefficient (MCC)*. The results are presented through two separate heatmaps, corresponding to the validation and test datasets.

Test 6 The performance results for Test 6 are summarized in Figure 6.13. On the *Validation* set, it is evident that the *GMM* and *AE* models achieve perfect identification of all anomalies. It can also be observed that all models obtain an *MCC* value of zero, as every AP in this dataset exhibits at least one anomaly. Within this set, the *IF* model shows the worst performance across all metrics except for *Precision*.

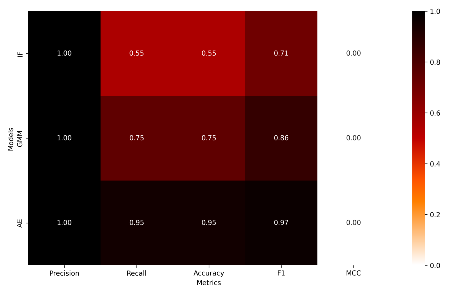
On the *Test* set, the *IF* model again performs the worst, presenting the lowest *Recall* value and consequently the highest *FNR* (equal to 0.36). The other two models successfully identify all anomalies in this dataset, achieving a *Recall* of 1.0. Notably, *GMM* and *AE* perform identically across all metrics on this set, although both generate some False Anomalies (with a *Precision* of 0.73).

Overall, in this test, the *GMM* and *AE* models outperform *IF* in terms of *FNR*; however, both also introduce several False Anomalies in the test dataset.

Test 7 The results of Test 7 are presented in Figure 6.14. For both the *Validation* and *Test* sets, all models obtain an *MCC* value of 0.0, a consequence of the fact that every AP in this dataset exhibits at least one anomaly. From the *Validation* set results, it is clear that *AE* is the only model capable of identifying



(a) Validation.



(b) Test.

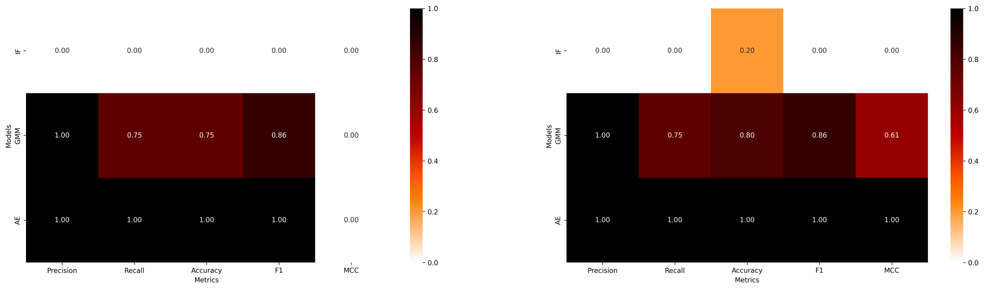
Figure 6.14: Performance of the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) in Test 7 of the anomaly detection task. For each model, the evaluation metrics include *Precision*, *Recall*, *Accuracy*, *F1-Score*, and *Matthews Correlation Coefficient (MCC)*. The results are presented through two separate heatmaps, corresponding to the validation and test datasets.

all anomalies. The *IF* model, by contrast, performs worst on this set, showing the lowest *Recall* value and consequently the highest *FNR* (0.60). Thus, on this dataset, *IF* is the poorest-performing model, while *AE* is the best.

For the *Test* set, the *AE* model once again demonstrates the strongest performance, achieving the highest values across all metrics and presenting an *FNR* of 0.05. The *IF* model remains the worst performer, obtaining the lowest values across all metrics, with a *Recall* of 0.55 and an *FNR* of 0.45.

In summary, across both datasets, the *AE* model consistently performs best, followed by *GMM*, and lastly *IF*.

Test 8 The results of Test 8 are presented in Figure 6.15. On the *Validation* set, it is evident that the *IF* model performs the worst, as it fails to identify any of the anomalies in the dataset and consequently obtains a value of 0.0 for all metrics. By contrast, the *AE* model is the only one achieving perfect anomaly identification, obtaining a value of 1.0 across all metrics and thus an *FNR* of 0.0. As in previous tests, all Access Points in the *Validation* set are anomalous, which results in an *MCC* value of zero for all models. It is also worth noting that the *GMM* model fails to detect some anomalies, achieving a *Recall* of 0.75.



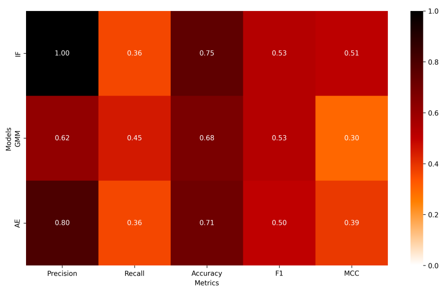
(a) Validation. **(b)** Test.

Figure 6.15: Performance of the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) in Test 8 of the anomaly detection task. For each model, the evaluation metrics include *Precision*, *Recall*, *Accuracy*, *F1-Score*, and *Matthews Correlation Coefficient (MCC)*. The results are presented through two separate heatmaps, corresponding to the validation and test datasets.

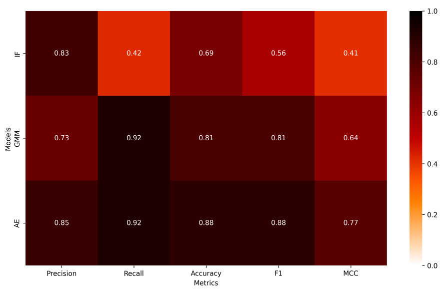
Regarding the *Test* set, *IF* again performs the worst, obtaining zero across all metrics and therefore a low *Accuracy* of 0.20. As before, the *AE* model is the only one achieving perfect classification, while the *GMM* model fails to identify certain anomalies, resulting in an *FNR* of 0.25. However, it should be noted that *GMM* does not produce any False Alarms, achieving a *Precision* value of 1.0.

Overall, in this test the *AE* model clearly outperforms the others, achieving perfect anomaly identification on both datasets, while the *IF* model yields the poorest results.

Test 9 The results of Test 9 are summarized in Figure 6.16. On the *Validation* set, the *IF* model is the only one that does not generate False Alarms, achieving a *Precision* of 1.0. Meanwhile, the *GMM* model produces the lowest number of missed anomalies, achieving a *Recall* of 0.45, which leads it to achieve the best *F1-Score*. The *AE* model, in turn, obtains the highest *MCC* value (0.39). Overall, all models perform the task poorly on this dataset, exhibiting excessively high *FNR* values.



(a) Validation.



(b) Test.

Figure 6.16: Performance of the Isolation Forest (IF), Gaussian Mixture Model (GMM), and Autoencoder (AE) in Test 9 of the anomaly detection task. For each model, the evaluation metrics include *Precision*, *Recall*, *Accuracy*, *F1-Score*, and *Matthews Correlation Coefficient (MCC)*. The results are presented through two separate heatmaps, corresponding to the validation and test datasets.

On the *Test* set, the models perform noticeably better, identifying anomalies more reliably. All models generate False Alarms (as indicated by *Precision* values below 1.0) and all fail to detect at least one anomaly (*FNR* greater than zero). However, there is a substantial performance gap between *IF* and the other two models, as *IF* achieves the lowest scores across all metrics. Overall, the *AE* model appears to be the best performer on this dataset, obtaining the highest values across all metrics.

In summary, this test highlights that none of the models successfully identifies anomalies to a satisfactory degree on the *Validation* set. However, on the *Test* set, the *AE* model provides the best performance among the three.

6.6 Feature Selection

In this section, we present the performance results of the three anomaly detection models across all the tests described in section 6.5. Each test was conducted using the feature subsets selected by the four Feature Selection Methods introduced in section 6.6. The specific features selected for each backbone are summarized in Appendix B.

Test 1 The results obtained on the datasets of Test 1 are presented in Figure 6.17. For the *IF* and *GMM* models, it is evident that, on the validation set, the *DET* feature selection method yields the poorest performance across all metrics, resulting in the highest number of missed anomalies. Conversely, for all three anomaly detection models, the *PCA* method consistently provides the best performance on the validation set.

Interestingly, for the *IF* model, the relative performance of the feature selection methods appears inverted between the validation and test sets. Specifically, *DET* produces the best results on the test set but the worst on the validation set for *IF*. A similar inversion is observed for *K-Means* and *PCA*, which yield strong results on the test set but comparatively weaker performance on the validation set.

For the *GMM* model, the feature selection methods that perform poorly on the validation set (*DET* and *K-Means*) also exhibit poor performance on the test set. In contrast, *PCA* and *CFS* lead to the best overall results across both sets and metrics.

Regarding the *AE* model, the *CFS* method performs the worst across all metrics in both the validation and test sets, followed by *DET*. In this case, *PCA* and *K-Means* yield the best results for the *AE* model.

Overall, *PCA* emerges as the feature selection method that most consistently delivers strong performance across all metrics, for both the validation and test sets, and for all three anomaly detection models. In contrast, the *DET* method appears less stable, tending to produce poor performance for *GMM* and suboptimal results for *AE*.

Test 2 The results obtained for Test 2 are summurized in Figure 6.18. For *Isolation Forest* (IF), performance remains highly dependent on the feature selection method. The *PCA*-based configuration achieves strong validation results but exhibits a noticeable degradation on the test set, particularly due to an increase in the False Negative Rate (0.25) and a corresponding drop in F1 Score (0.67). In contrast, the *DET*, *CFS*, and *K-Means* subsets yield perfect recall (1.00) on the validation set, though their precision descreases significantly. Among these, *DET* emerges as the most balanced method in the test set, achieving the highest F1 Score (0.89) and MCC (0.88), whereas *CFS* consistently delivers the weakest validation performance for IF, with a precision of only 0.20 and an F1 Score of 0.33.

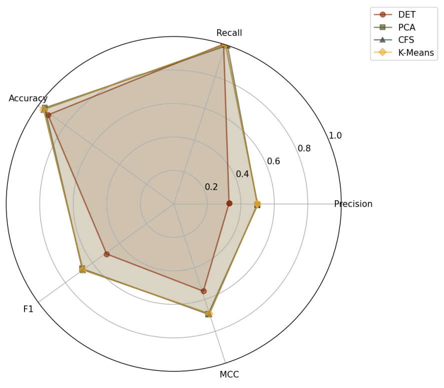
The *Gaussian Mixture Model* (GMM) displays a markedly more stable behavior across feature selection methods. Both *PCA* and *K-Means* yield perfect validation performance (all metrics equal to 1.00), and although their recall drops to 0.75 in the test set, they maintain high precision (1.00) and strong F1 Scores (0.86). Conversely, the *CFS* method produces the least effective results for GMM, with the lowest recall (0.50) and F1 Score (0.67) in both validation and test sets. The *DET* subset performs reasonably well, offering a competitive performance, though it does not reach the levels achieved by PCA or K-Means on validation set.

For what concerns the *Autoencoder* (AE), both *PCA* and *K-Means* produce perfect performance in validation and test sets. The *CFS* and *DET* methods also yield optimal results on the validation set, although their test performance shows a slight decrease in recall (0.75), leading to F1 Scores of 0.86. Despite this reduction, AE maintains high robustness, outperforming IF and GMM across all feature selection methods.

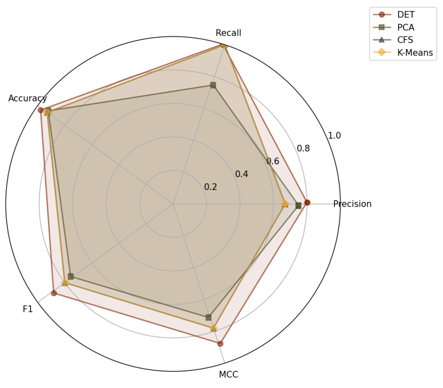
Overall, the radar plots reveal clear trends in the relative effectiveness of the feature selection techniques. *PCA* and *K-Means* consistently emerge as the best-performing methods, especially when paired with GMM and, most notably, AE. In contrast, *CFS* proves to be the least reliable technique across all models, resulting in lower recall and reduced F1 Scores.

Test 3 In Test 3 (Figure 6.19), the radar plots reveal substantial differences emerging across feature selection methods.

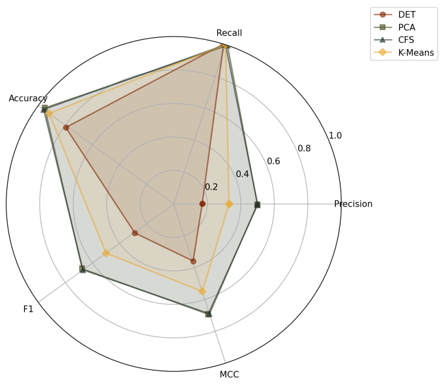
Isolation Forest (IF) exhibits the weakest performance among the three



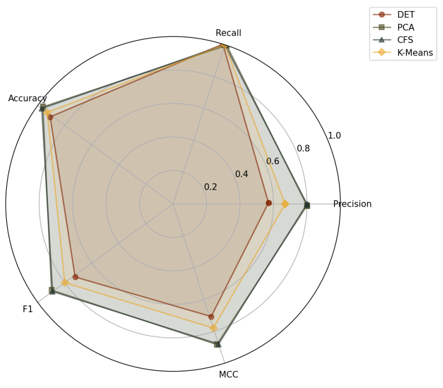
(a) IF on Val.



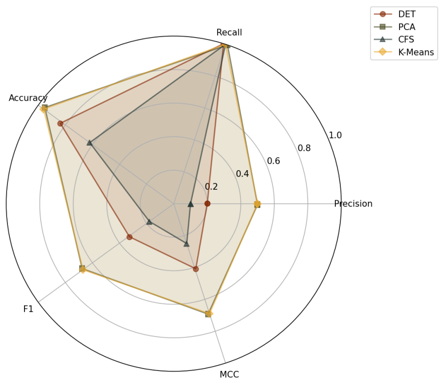
(b) IF on Test.



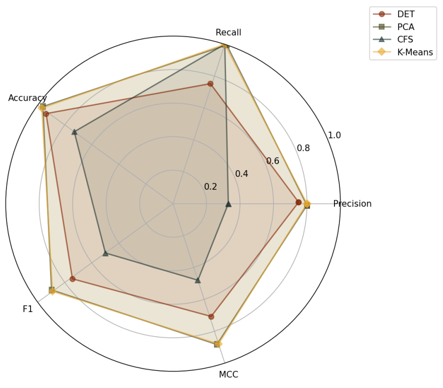
(c) GMM on Val.



(d) GMM on Test.

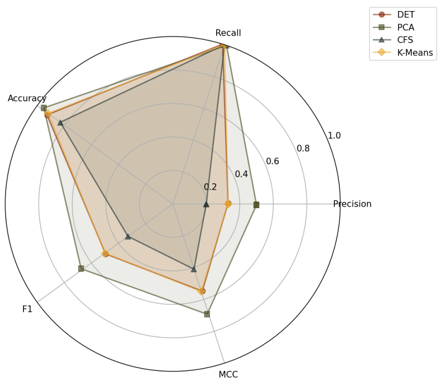


(e) AE on Val.

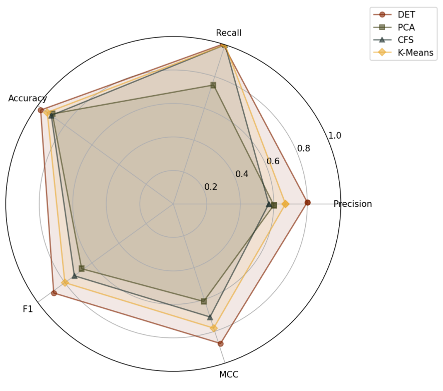


(f) AE on Test.

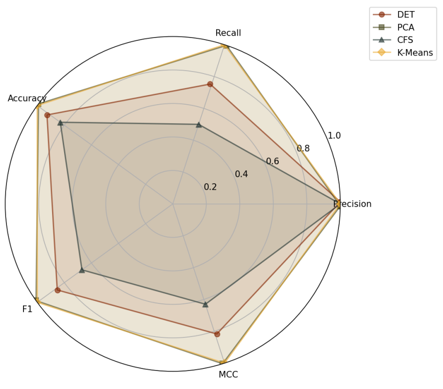
Figure 6.17: Radar plots comparing the performance of *Isolation Forest (IF)*, *Gaussian Mixture Model (GMM)*, and *Autoencoder (AE)* on the Validation (Val) and Test sets generated for TEST 1. Each plot illustrates the metrics obtained using feature subsets derived from four Feature Selection Methods: *Domain Expert Technique (DET)*, *Principal Component Analysis (PCA)*, *Correlation-Based Feature Selection (CFS)*, and *K-Means Clustering-Based Feature Selection (K-Means)*.



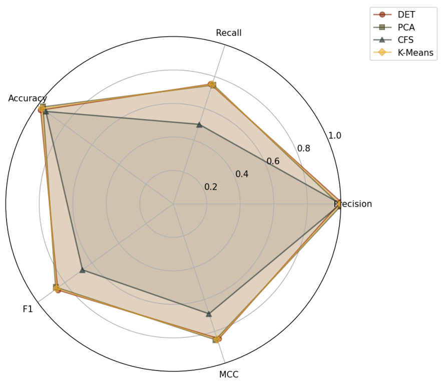
(a) IF on Val.



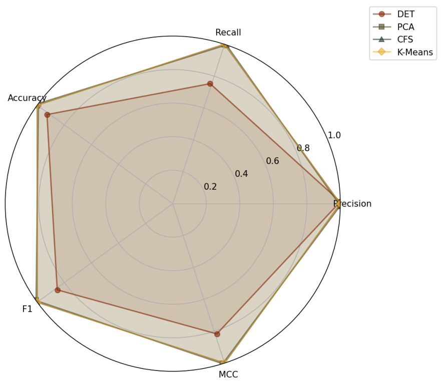
(b) IF on Test.



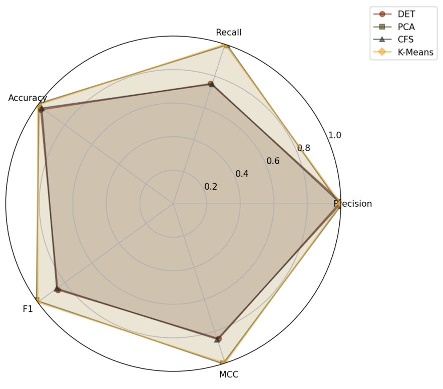
(c) GMM on Val.



(d) GMM on Test.



(e) AE on Val.



(f) AE on Test.

Figure 6.18: Radar plots comparing the performance of *Isolation Forest (IF)*, *Gaussian Mixture Model (GMM)*, and *Autoencoder (AE)* on the Validation (Val) and Test sets generated for TEST 2. Each plot illustrates the metrics obtained using feature subsets derived from four Feature Selection Methods: *Domain Expert Technique (DET)*, *Principal Component Analysis (PCA)*, *Correlation-Based Feature Selection (CFS)*, and *K-Means Clustering-Based Feature Selection (K-Means)*.

models, with all feature selection methods yielding limited recall, especially in the validation set where values range from 0.09 (PCA) to 0.27 (K-Means). Despite consistently perfect precision (1.00) across all methods, the high False Negative Rates (low Recall) severely reduce F1 Scores and MCC values. Among the available feature subsets, *K-Means* emerges as the most effective method for IF, achieving the highest validation F1 Score (0.43) and the strongest test performance (F1 Score = 0.67, MCC = 0.59). Conversely, *PCA* leads to the poorest results for IF, particularly in validation (F1 Score = 0.17), indicating that PCA-based features are insufficiently informative for this model under the conditions of Test 3.

The *Gaussian Mixture Model* (GMM) demonstrates markedly better performance than IF, with a clear and consistent hierarchy among feature selection methods. The *K-Means* subset delivers the strongest validation performance (F1 Score = 0.84, MCC = 0.79), followed by *PCA* and *CFS*, which achieve comparable results (F1 Score = 0.71). These trends persist in test performance, where *CFS* yields the highest F1 Score (0.86), while *K-Means* and *PCA* remain strong and stable. In contrast, the *DET* subset consistently produces the weakest outcomes for GMM, with very low recall (0.18–0.17) and F1 Scores around 0.30.

The *Autoencoder* (AE) achieves the most balanced results in Test 3, outperforming both IF and GMM across nearly all feature subsets. While validation performance varies substantially across methods, AE maintains consistently good test performance. The *CFS* method provides the best validation results for AE (F1 Score = 0.78, MCC = 0.72), and also performs strongly on the test set (F1 Score = 0.80). *DET* also shows solid test-set generalization (F1 Score = 0.80), despite weaker validation performance. The *PCA* subset yields moderate validation metrics but strong test results (F1 Score = 0.83). Conversely, *K-Means* is the least effective method for AE, particularly in the test set (F1 Score = 0.67).

Thus, *K-Means* emerges as the most effective method for GMM, and the only one that offers competitive performance for IF. For AE, however, *CFS* and *PCA* provide the best overall performance. Across all models, *DET* consistently ranks among the weakest feature selection strategies.

Test 4 Observing the results of Test 4 (Figure 6.20), the radar plots reveal that the three anomaly detection models exhibit highly heterogeneous performance.

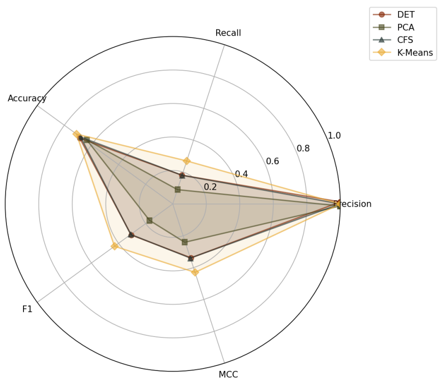
Isolation Forest (IF) performs the weakest among the three models, with extremely poor validation results across most feature selection methods. In particular, the *DET*, *PCA*, and *K-Means* subsets all lead to complete detection failure in the validation set (Recall = 0.00), indicating that these feature representations provide insufficient discriminatory information for IF in this test. The *CFS* subset yields slightly better validation performance (Recall = 0.11), but still far below acceptable levels. On the test set, IF exhibits moderate improvements, particularly with *K-Means*, which achieves the highest performance for this model (F1 Score = 0.70; MCC = 0.59). *CFS* also performs reasonably (F1 Score = 0.50). Nevertheless, the stark contrast between validation and test performance across methods indicates that IF is highly unstable and sensitive to feature representation in Test 4. Among IF's feature subsets, *K-Means* can be considered the least ineffective, while *DET* and *PCA* clearly perform the worst.

The *Gaussian Mixture Model* (GMM) shows significantly stronger and more stable behaviour than IF. In the validation set, the best-performing methods are *DET* (F1 Score = 0.62) and *PCA* (0.63), followed by *CFS*. Interestingly, although *K-Means* yields relatively weaker validation metrics (F1 Score = 0.53), it achieves the strongest performance on the test set, with near-perfect anomaly detection (F1 Score = 0.97; MCC = 0.93; Recall = 1.00). This suggests that the K-Means-selected features generalize exceptionally well despite modest validation performance. The *DET* subset also exhibits excellent test performance (F1 Score = 0.90), contrasting with weaker validation results. *CFS* and *PCA* show solid and consistent behaviour across both datasets. Overall, for GMM, *K-Means* emerges as the best-performing feature selection method in Test 4, while *CFS* and *PCA* remain reliable alternatives. The weakest subset for GMM in the test set is the *CFS* validation performance (F1 Score = 0.50; Recall = 0.33), but even this method performs reasonably well once evaluated on the test set.

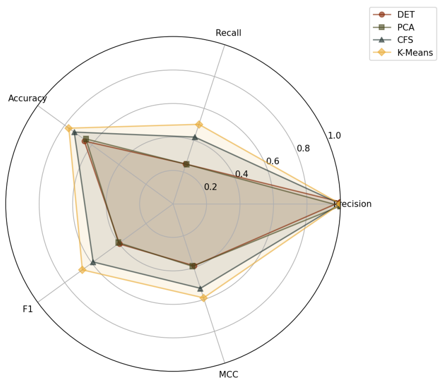
The *Autoencoder* (AE) demonstrates the most stable performance across models in Test 4. All feature selection methods yield moderate-to-strong validation performance, with *CFS* providing the highest validation F1 Score (0.67), followed closely by *DET* (0.62). Although *PCA* and *K-Means* exhibit slightly weaker validation performance, they deliver outstanding generalization on the test set. In particular, *PCA* achieves near-perfect detection in the test set (F1 Score = 0.97; MCC = 0.93), closely followed by *K-Means* (F1 Score = 0.93; MCC = 0.86). The *CFS* subset also performs strongly (F1 Score = 0.87). Also, the *DET* method seems to be good (F1 Score = 0.82), though not at the level of PCA or K-Means. Across all feature subsets, AE consistently outperforms IF and performs comparably or better than GMM.

Collectively, the results of Test 4 highlight significant differences in the suitability of feature selection strategies across anomaly detection models. *K-Means* emerges as the strongest overall method for GMM and the most competitive option for IF, while *PCA* and *CFS* deliver the best and most stable performance for AE. In contrast, *DET* frequently ranks among the weaker-performing methods, particularly for IF, although it still provides strong results for GMM and AE in the test set.

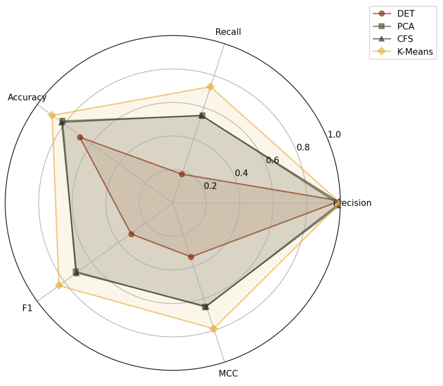
Test 5 The results obtained for Test 5 (Figure 6.21) reveal that overall, PCA consistently emerges as the most reliable method, whereas CFS and DET often yield suboptimal performance depending on the model and split.



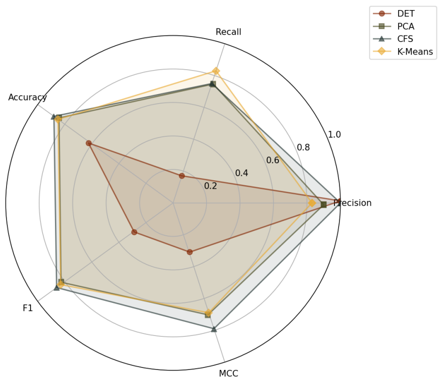
(a) IF on Val.



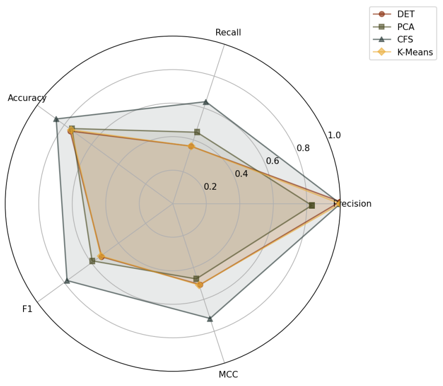
(b) IF on Test.



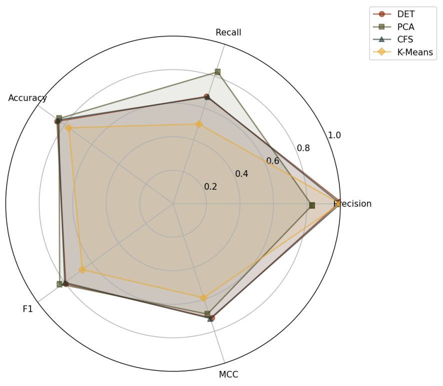
(c) GMM on Val.



(d) GMM on Test.

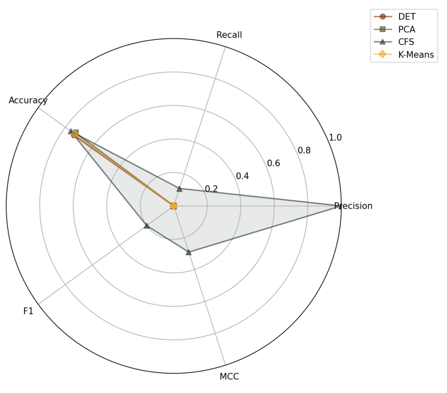


(e) AE on Val.

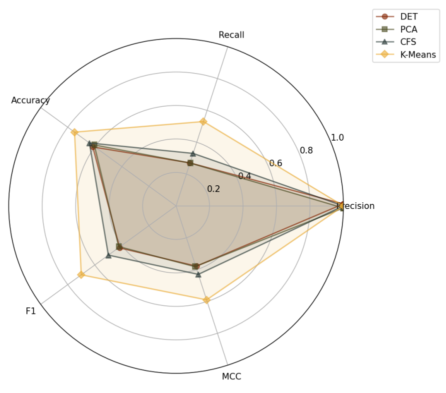


(f) AE on Test.

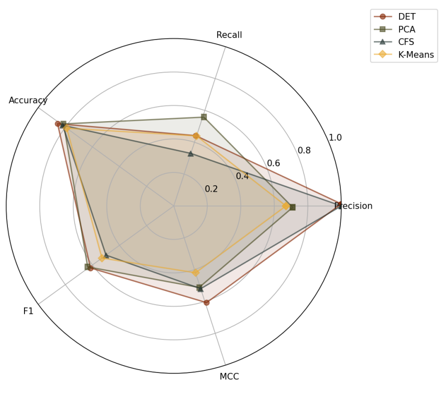
Figure 6.19: Radar plots comparing the performance of *Isolation Forest (IF)*, *Gaussian Mixture Model (GMM)*, and *Autoencoder (AE)* on the Validation (Val) and Test sets generated for TEST 3. Each plot illustrates the metrics obtained using feature subsets derived from four Feature Selection Methods: *Domain Expert Technique (DET)*, *Principal Component Analysis (PCA)*, *Correlation-Based Feature Selection (CFS)*, and *K-Means Clustering-Based Feature Selection (K-Means)*.



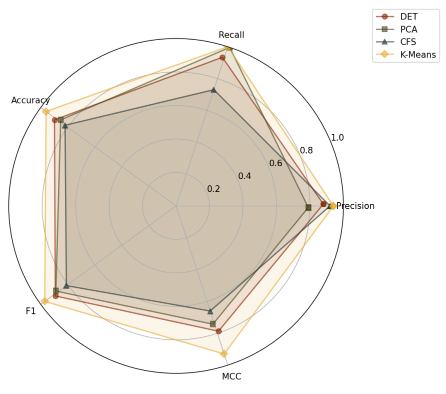
(a) IF on Val.



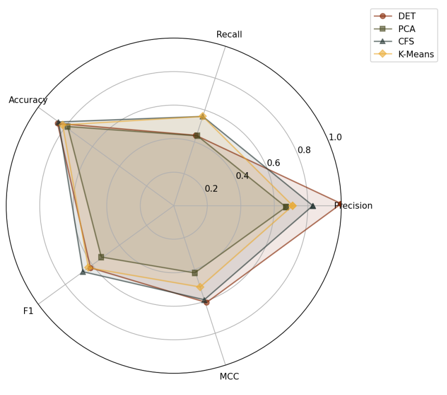
(b) IF on Test.



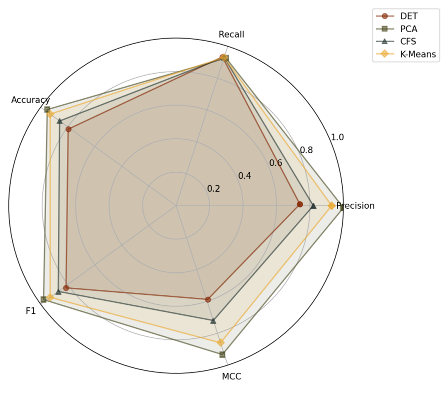
(c) GMM on Val.



(d) GMM on Test.



(e) AE on Val.



(f) AE on Test.

Figure 6.20: Radar plots comparing the performance of *Isolation Forest (IF)*, *Gaussian Mixture Model (GMM)*, and *Autoencoder (AE)* on the Validation (Val) and Test sets generated for TEST 4. Each plot illustrates the metrics obtained using feature subsets derived from four Feature Selection Methods: *Domain Expert Technique (DET)*, *Principal Component Analysis (PCA)*, *Correlation-Based Feature Selection (CFS)*, and *K-Means Clustering-Based Feature Selection (K-Means)*.

Considering the Isolation Forest model, DET provides the strongest performance on the test set, achieving perfect Recall (1.00), F1 Score (1.00), and MCC (1.00), with an FNR of 0.00, thereby detecting all anomalies without misclassification. However, on the validation set its performance degrades substantially (Recall 0.80, F1 Score 0.73, MCC 0.65, FNR 0.20), remaining the best feature selection method in terms of Recall performances. PCA yields the lowest validation performance among all methods (Recall 0.20, F1 Score 0.33, MCC 0.40, FNR 0.80), driven by a very high FNR, yet performs competitively on the test set (Recall 0.75, F1 Score 0.86, MCC 0.85, FNR 0.25). K-Means delivers strong and stable results across both splits, achieving perfect detection on the test set (Recall 1.00, F1 Score 1.00, MCC 1.00, FNR 0.00) and the highest validation performance among the four methods (Recall 0.60, F1 Score 0.75, MCC 0.73, FNR 0.40) in terms of all metrics, except for Recall. In contrast, CFS provides only moderate validation performance (Recall 0.40, F1 Score 0.50, MCC 0.42, FNR 0.60) and matches PCA on the test set (Recall 0.75, F1 Score 0.86, MCC 0.85, FNR 0.25).

Across the GMM model, PCA is clearly the best-performing method, achieving perfect results on the validation set (Recall 1.00, F1 Score 1.00, MCC 1.00, FNR 0.00) and near-perfect outcomes on the test set (Recall 1.00, F1 Score 0.89, MCC 0.88, FNR 0.00). K-Means is the second-best option, with high validation performance (Recall 1.00, F1 Score 0.91, MCC 0.89, FNR 0.00) and good test results (Recall 1.00, F1 Score 0.89, MCC 0.88, FNR 0.00). DET performs adequately (validation Recall 0.80, F1 Score 0.80, MCC 0.74, FNR 0.20), but noticeably worse than PCA and K-Means. CFS, instead, provides the weakest validation performance (Recall 0.20, F1 Score 0.33, MCC 0.40, FNR 0.80), again driven by a very high FNR, although its test performance improves substantially (Recall 0.75, F1 Score 0.86, MCC 0.85, FNR 0.25).

For the Autoencoder, PCA is again the most effective method, achieving perfect performance on the test set (Recall 1.00, F1 Score 1.00, MCC 1.00, FNR 0.00) and strong results on the validation set (Recall 0.80, F1 Score 0.80, MCC 0.74, FNR 0.20). DET and CFS provide identical validation results (Recall 0.80, F1 Score 0.80, MCC 0.74, FNR 0.20), but DET underperforms on the test set (Recall 0.75, F1 Score 0.67, MCC 0.62, FNR 0.25). K-Means yields the weakest validation performance for the AE model (Recall 0.60, F1 Score 0.67, MCC 0.59, FNR 0.40), although its test performance remains competitive (Recall 1.00, F1 Score 0.89, MCC 0.88, FNR 0.00).

Overall, PCA stands out as the most consistently effective feature–selection method across all three models and both dataset splits. Conversely, CFS and DET exhibit notable instability, with CFS frequently suffering from extremely high FNR values (up to 0.80), indicating a substantial number of missed anomalies, and DET demonstrating significant fluctuations between validation and test sets. K-Means generally performs well, particularly for GMM and AE, but PCA remains the only method that provides uniformly strong and reliable performance across all evaluation metrics.

Test 6 The results of Test 6 (Figure 6.22) reveal substantial variability in the effectiveness of different feature–selection methods across all anomaly–detection models. Overall, the Isolation Forest model exhibits considerable difficulty on this dataset regardless of the selected features, while GMM and AE models achieve markedly stronger results, particularly when paired with PCA, CFS, or K-Means.

For the Isolation Forest, all feature–selection methods perform poorly on the validation set, yielding identical Recall (0.33), F1 Score (0.50), MCC (0.00), and FNR (0.67). This indicates that IF fails to correctly identify two thirds of anomalies independently of the selected features. On the test set, CFS achieves the strongest performance, with Recall 0.64, F1 Score 0.78, MCC 0.68, and a lower FNR of 0.36, making it the only method enabling the model to recover a meaningful portion of anomalies. K-Means follows with Recall 0.55, F1 Score 0.71, MCC 0.61, and FNR 0.45. PCA yields moderate results (Recall 0.45, F1 Score 0.62, MCC 0.54, FNR 0.55), while DET performs worst, with Recall 0.18, F1 Score 0.31, MCC 0.32, and an FNR as high as 0.82. In summary, IF suffers from severe recall degradation across all methods, but CFS provides the comparatively most effective feature subset.

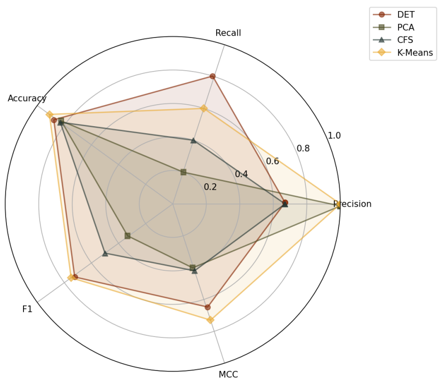
For the GMM model, all feature–selection methods achieve perfect validation performance (Recall 1.00, F1 Score 1.00, FNR 0.00), although the MCC remains 0.00 due to class imbalance. On the test set, K-Means yields the strongest overall results with Recall 1.00, F1 Score 0.85, MCC 0.68, and FNR 0.00. CFS and PCA follow closely, both achieving perfect Recall and FNR 0.00, with F1 Scores of 0.81 and 0.79 and MCC values of 0.61 and 0.54, respectively. DET performs slightly worse (Recall 0.82, F1 Score 0.82, MCC 0.64, FNR 0.18). The consistently high Recall and low FNR across PCA, CFS, and K-Means indicate that test performance differences are primarily driven by precision variability, making K-Means the most stable and discriminative method for this model.

The Autoencoder model shows similar behaviour, with CFS and K-Means achieving perfect validation scores (Recall 1.00, F1 Score 1.00, FNR 0.00), while DET and PCA also reach strong validation performance (Recall 0.67, F1 Score 0.80, FNR 0.33). On the test set, CFS provides the best results overall, with balanced and high metrics (Recall 0.91, F1 Score 0.91, MCC 0.82, FNR 0.09), making it the most reliable feature–selection method for AE in this test. PCA also performs well, achieving perfect Recall (1.00), an FNR of 0.00, F1 Score 0.88, and MCC 0.76. DET follows with Recall 0.73, F1 Score 0.84, MCC 0.76, and FNR 0.27. K-Means yields weaker test performance despite perfect validation scores, achieving Recall 1.00 but lower F1 Score 0.76 and MCC 0.47 due to reduced precision (0.61).

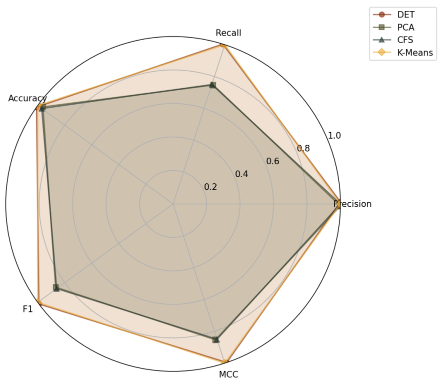
Overall, Test 6 highlights a clear separation in feature–selection methods effectiveness. PCA, CFS, and K-Means consistently provide strong performance for GMM and AE, with CFS emerging as the most robust method due to its ability to maintain high Recall (above 0.90) and low FNR (down to 0.00) across multiple models. K-Means also performs strongly, particularly for GMM, where it offers the highest F1 Score and MCC on the test set. In contrast, DET proves unstable across all models, frequently producing higher FNR values (up to 0.82) and substantially lower Recall, making it the least reliable feature–selection strategy in this test.

Test 7 The results of Test 7 (Figure 6.23) indicate a clear separation in performance across feature–selection methods and models. The Isolation Forest model performs poorly overall, with all feature–selection methods suffering from extremely high FNRs and limited anomaly detection capability. DET and CFS show the weakest behaviour, both achieving a validation Recall of 0.00, F1 Score of 0.00, and a FNR of 1.00, revealing complete failure to detect anomalies. Their test performance remains severely degraded, as DET obtains Recall 0.30, F1 Score 0.46, and an FNR of 0.70, while CFS yields an even lower Recall of 0.10, F1 Score 0.18, and FNR 0.90, confirming that these feature subsets lead the model to miss the vast majority of anomalies. PCA and K-Means represent comparatively better choices for IF, although performance remains limited. Both achieve validation Recall 0.20, F1 Score 0.33, FNR 0.80, and identical test performance with Recall 0.45, F1 Score 0.62, and FNR 0.55. These results indicate that PCA and K-Means enable the model to identify less than half of the anomalies and are therefore only marginally better than DET and CFS. Overall, Isolation Forest remains the worst model in this test.

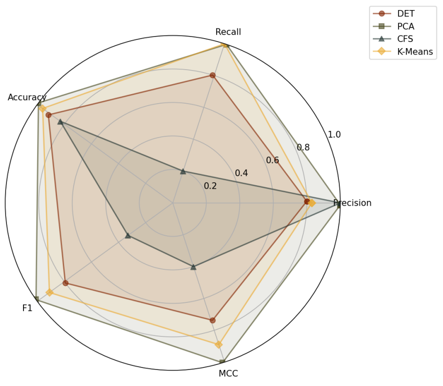
In contrast, the GMM model demonstrates robust and stable performance across all feature–selection methods. PCA constitutes the best feature–selection method for GMM, achieving perfect validation scores (Recall 1.00, F1 Score 1.00, FNR 0.00) and excellent test performance with Recall 0.85, F1 Score 0.92, and an FNR of only 0.15, confirming its ability to retain highly discriminative information. K-Means also performs strongly, with validation Recall 0.80, F1 Score 0.89, FNR 0.20, and test Recall 0.75, F1 Score 0.86, FNR 0.25; its slight reduction in precision–related metrics compared to PCA reflects minor



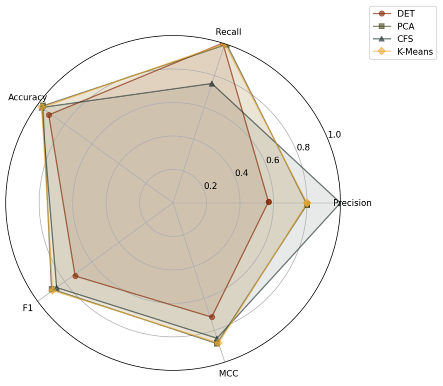
(a) IF on Val.



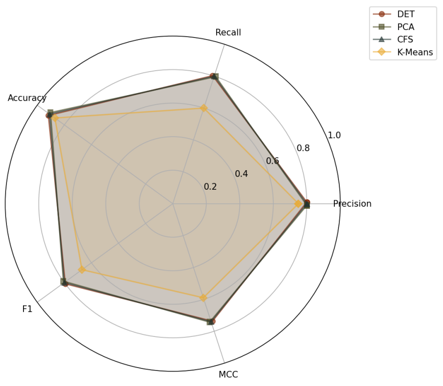
(b) IF on Test.



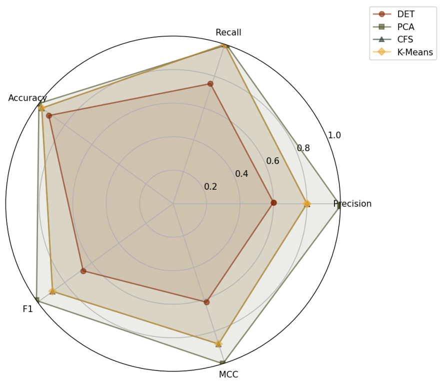
(c) GMM on Val.



(d) GMM on Test.

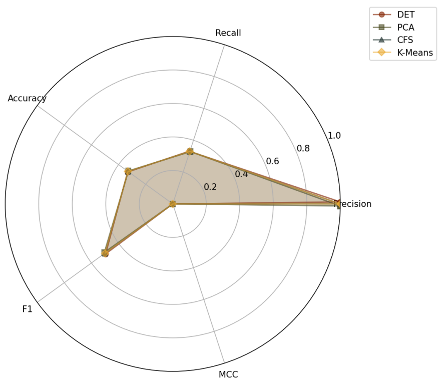


(e) AE on Val.

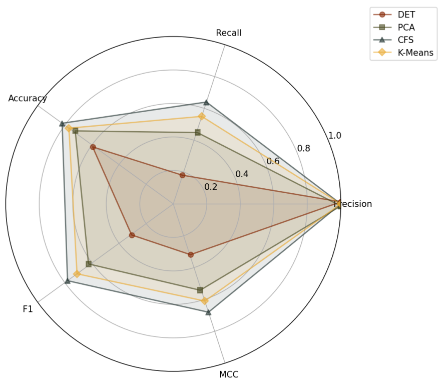


(f) AE on Test.

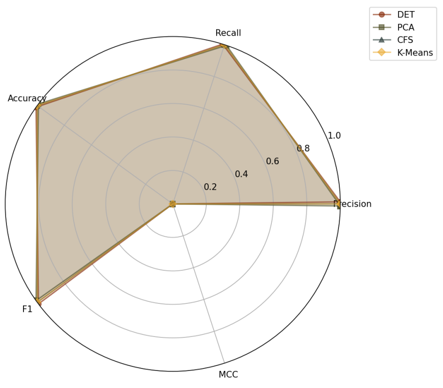
Figure 6.21: Radar plots comparing the performance of *Isolation Forest (IF)*, *Gaussian Mixture Model (GMM)*, and *Autoencoder (AE)* on the Validation (Val) and Test sets generated for TEST 5. Each plot illustrates the metrics obtained using feature subsets derived from four Feature Selection Methods: *Domain Expert Technique (DET)*, *Principal Component Analysis (PCA)*, *Correlation-Based Feature Selection (CFS)*, and *K-Means Clustering-Based Feature Selection (K-Means)*.



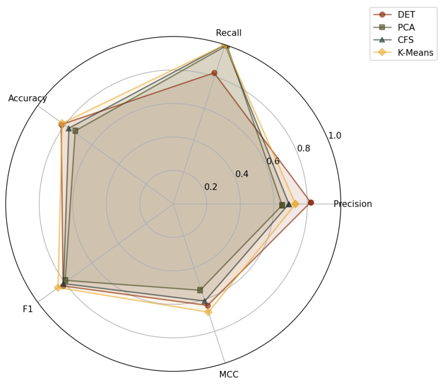
(a) IF on Val.



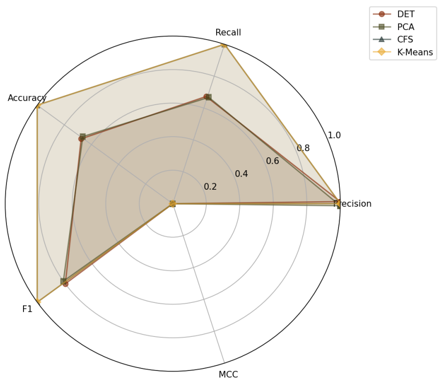
(b) IF on Test.



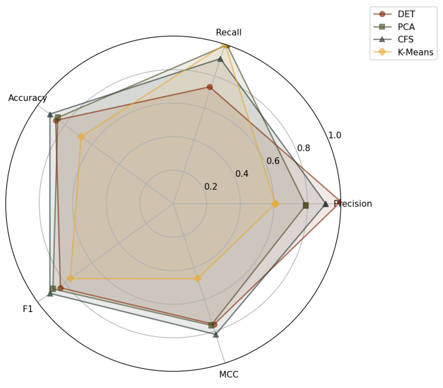
(c) GMM on Val.



(d) GMM on Test.



(e) AE on Val.



(f) AE on Test.

Figure 6.22: Radar plots comparing the performance of *Isolation Forest (IF)*, *Gaussian Mixture Model (GMM)*, and *Autoencoder (AE)* on the Validation (Val) and Test sets generated for TEST 6. Each plot illustrates the metrics obtained using feature subsets derived from four Feature Selection Methods: *Domain Expert Technique (DET)*, *Principal Component Analysis (PCA)*, *Correlation-Based Feature Selection (CFS)*, and *K-Means Clustering-Based Feature Selection (K-Means)*.

over–generalisation. DET and CFS provide moderate but consistent results, each yielding validation Recall 0.60, F1 Score 0.75, FNR 0.40, and test Recall values of 0.75 and 0.70, respectively, with test F1 Scores of 0.86 and 0.82.

The Autoencoder exhibits performance trends that closely mirror those observed in GMM, although with stronger results overall. PCA again emerges as a top feature–selection method, achieving perfect validation scores (Recall 1.00, F1 Score 1.00, FNR 0.00) and high test performance (Recall 0.85, F1 Score 0.92, FNR 0.15). K-Means surpasses all other methods and all previously observed results in this test, obtaining validation Recall 0.80, F1 Score 0.89, FNR 0.20, and test Recall 0.95, F1 Score 0.97, and a low FNR of 0.05. These results indicate that the K-Means feature subset enables the AE to detect nearly all anomalies while maintaining perfect precision. DET and CFS display moderate but reliable performance, each achieving validation Recall 0.60, F1 Score 0.75, FNR 0.40 and test Recall values of 0.70 and 0.75, with corresponding F1 Scores of 0.82 and 0.86. Their higher FNRs (0.25–0.30) compared to PCA and K-Means show that they cause the model to miss a larger fraction of anomalies.

Overall, Test 7 confirms that PCA and K-Means are the two most effective feature–selection strategies, particularly for GMM and AE. DET and CFS consistently underperform compared to PCA and K-Means, especially in the Isolation Forest model where FNR values reach 1.00.

Test 8 For what concerns the results of Test 8, summurized in Figure 6.24, these reveal an extreme divergence in performance across the three anomaly detection models, with Isolation Forest exhibiting complete failure under every feature–selection method, while GMM and AE achieve near-perfect or fully perfect detection capabilities in most configurations. Isolation Forest represents the weakest model by a substantial margin: all four feature–selection methods (DET, PCA, CFS, K-Means) yield a validation Recall of 0.00, F1 Score of 0.00, and an FNR of 1.00, indicating that the model fails to detect any anomalies. The test results reproduce the same behaviour, with Recall 0.00, F1 Score 0.00, and an unchanged FNR of 1.00 for all methods despite a nominal Accuracy of 0.20. These results demonstrate that no feature subset enables IF to extract meaningful information from the Test 8 dataset, and that the model systematically misclassifies all anomalies regardless of the selected features.

In stark contrast, the Gaussian Mixture Model displays strong performance across all feature–selection methods, with especially remarkable results for the CFS method. CFS achieves perfect detection on both validation and test sets, obtaining Recall 1.00, F1 Score 1.00, and an FNR of 0.00, accompanied by an MCC of 1.00 on the test set. DET also performs exceptionally well, reaching perfect validation scores (Recall 1.00, F1 Score 1.00, FNR 0.00) and maintaining strong test performance, with Recall 0.75, F1 Score 0.86, and an FNR of 0.25. PCA and K-Means behave similarly, producing validation Recall 0.75, F1 Score 0.86, FNR 0.25, and identical test performance with Recall 0.75, F1 Score 0.86, and FNR 0.25 for both methods.

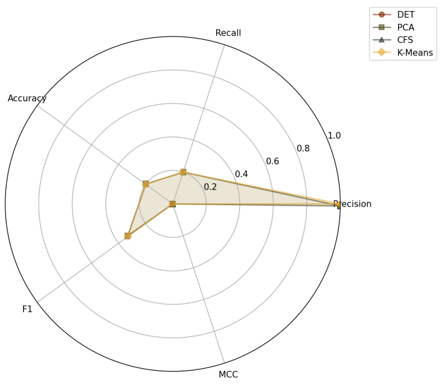
The Autoencoder follows the same trend as GMM but with even better results. DET, CFS, and K-Means all achieve perfect performance on both validation and test sets, each producing Recall 1.00, F1 Score 1.00, and an FNR of 0.00, alongside an MCC of 1.00 on the test set for all three methods. PCA also achieves perfect validation performance but registers a reduction on the test set, where Recall drops to 0.75, F1 Score to 0.86, and FNR rises to 0.25, mirroring the behaviour observed in GMM. Although PCA remains effective, its reduced Recall indicates that it may discard some discriminative information for AE in this test scenario. K-Means, CFS, and DET therefore constitute the strongest feature–selection methods for AE, as they consistently allow the model to achieve flawless anomaly detection across all metrics.

Overall, Test 8 confirms that the Isolation Forest model is entirely unable to handle this dataset regardless of the chosen feature–selection method, whereas both GMM and AE demonstrate good performance. The best-performing feature–selection method across all models is CFS, which achieves perfect detection for both GMM and AE on all sets. DET and K-Means also perform exceptionally well, particularly for AE, while PCA, although generally strong, proves slightly less reliable due to its higher FNR (0.25) and lower Recall (0.75) on the test sets of both GMM and AE. These results highlight that feature–selection methods such as CFS, DET, and K-Means effectively preserve anomaly-related structure in this dataset, enabling near-optimal or fully optimal detection in the more capable models.

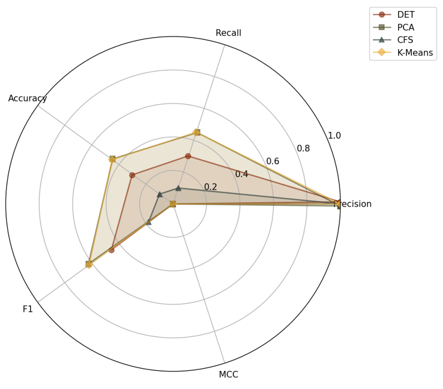
Test 9 The results obtained in Test 9 are shown in Figure 6.25, we can see that IF exhibits highly heterogeneous behaviour across feature sets: DET achieves moderate validation performance (Recall = 0.55, F1 = 0.71, MCC = 0.65) and improves substantially on the test set (Recall = 0.83, F1 = 0.80), indicating effective generalisation. CFS, in particular, provides the strongest performance for this model on the test set (F1 = 0.86, MCC = 0.79), but it is second in terms of Recall. Conversely, PCA leads to the weakest IF performance, with extremely high false negative rates (0.82 on validation and 0.67 on test) and correspondingly low Recall and F1 scores, indicating that PCA severely degrades the separability of anomalous samples for IF.

GMM-based models show similar patterns: both DET and PCA yield low validation Recall (0.27 and 0.18, respectively) and correspondingly low F1 scores. However, CFS and KMEANS perform substantially better. KMEANS on the test set achieves high recall (0.92), F1 score (0.81), and MCC (0.64). CFS also performs robustly (Recall = 0.83, F1 = 0.83, MCC = 0.69).

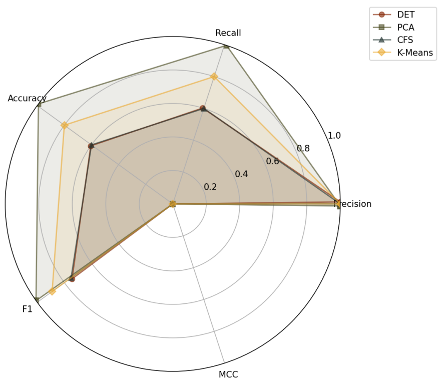
Autoencoders consistently outperform IF and GMM across nearly all settings. DET and PCA achieve excellent test performance (F1 = 0.91 and 0.88; MCC = 0.85 and 0.77, respectively). CFS and KMEANS, despite weaker validation performance, reach high test Recall (0.92 in both cases). Among all tested configurations, DET on the test set achieves the highest overall performance (Accuracy = 0.92, F1 = 0.91, MCC = 0.85), but not in terms of Recall and FNR.



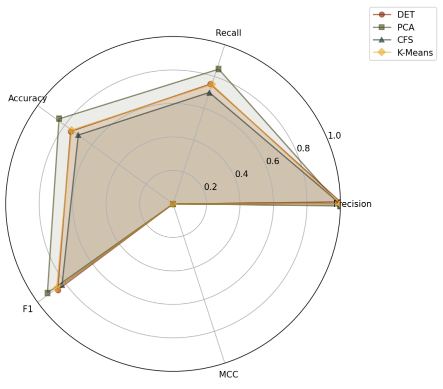
(a) IF on Val.



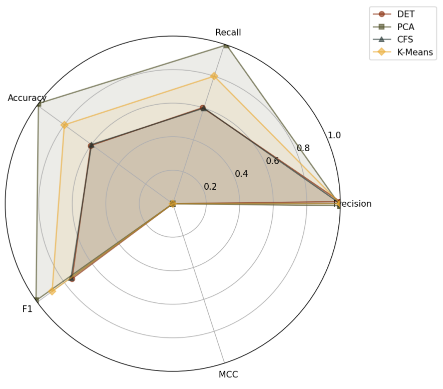
(b) IF on Test.



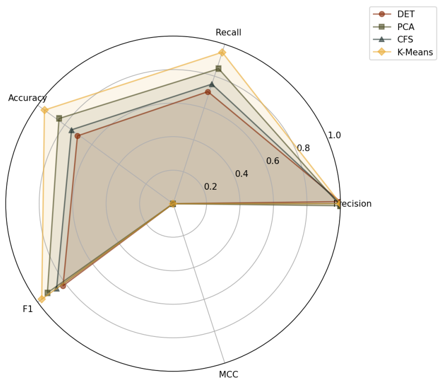
(c) GMM on Val.



(d) GMM on Test.

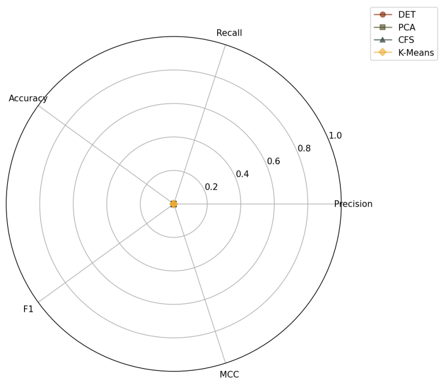


(e) AE on Val.

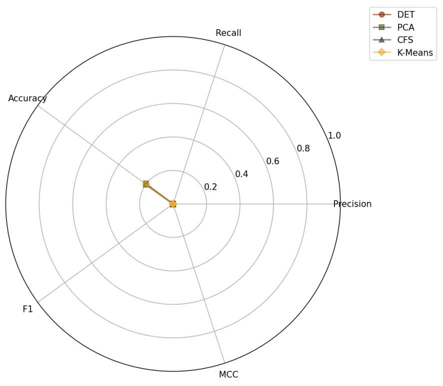


(f) AE on Test.

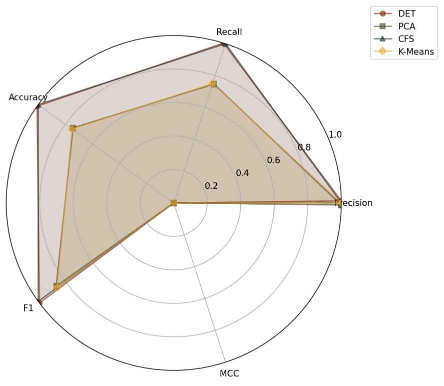
Figure 6.23: Radar plots comparing the performance of *Isolation Forest (IF)*, *Gaussian Mixture Model (GMM)*, and *Autoencoder (AE)* on the Validation (Val) and Test sets generated for TEST 7. Each plot illustrates the metrics obtained using feature subsets derived from four Feature Selection Methods: *Domain Expert Technique (DET)*, *Principal Component Analysis (PCA)*, *Correlation-Based Feature Selection (CFS)*, and *K-Means Clustering-Based Feature Selection (K-Means)*.



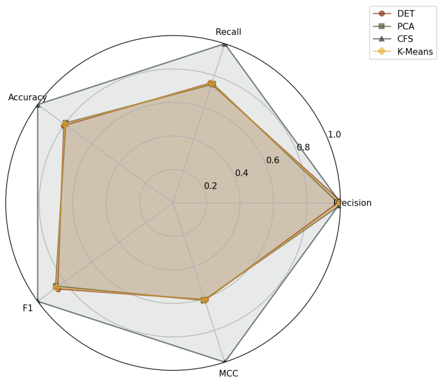
(a) IF on Val.



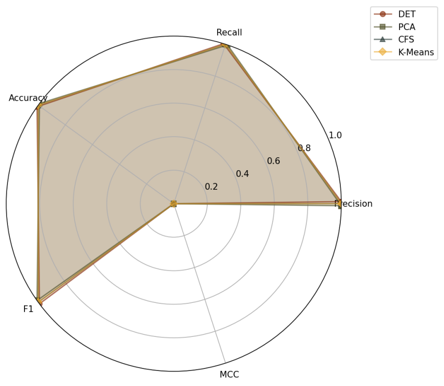
(b) IF on Test.



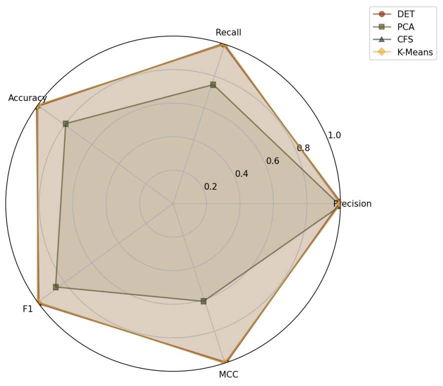
(c) GMM on Val.



(d) GMM on Test.



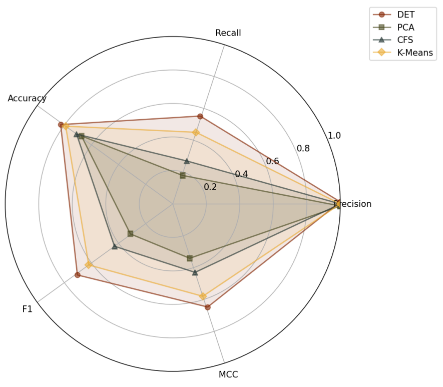
(e) AE on Val.



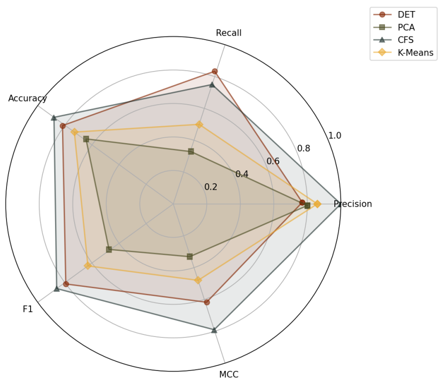
(f) AE on Test.

Figure 6.24: Radar plots comparing the performance of *Isolation Forest (IF)*, *Gaussian Mixture Model (GMM)*, and *Autoencoder (AE)* on the Validation (Val) and Test sets generated for TEST 8. Each plot illustrates the metrics obtained using feature subsets derived from four Feature Selection Methods: *Domain Expert Technique (DET)*, *Principal Component Analysis (PCA)*, *Correlation-Based Feature Selection (CFS)*, and *K-Means Clustering-Based Feature Selection (K-Means)*.

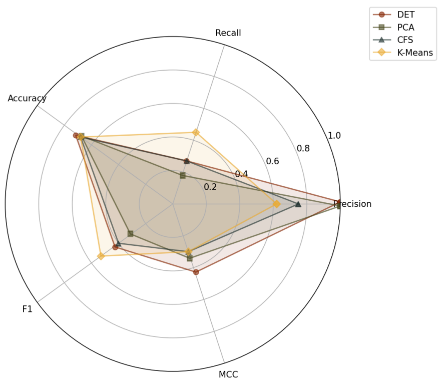
In summary, the results of Test 9 highlight two main conclusions. First, Autoencoders provide the most accurate and stable anomaly detection performance across feature sets. Secondly, PCA significantly degrades performance for IF and GMM.



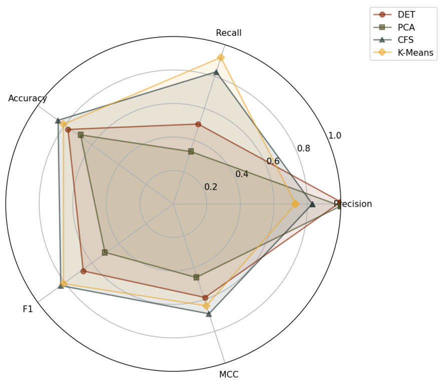
(a) IF on Val.



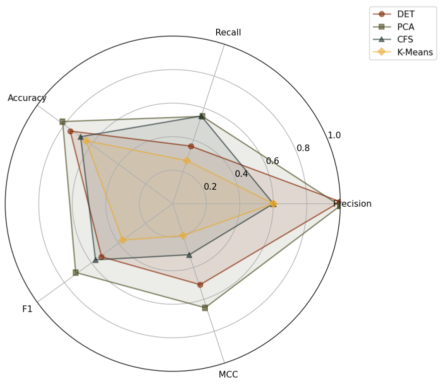
(b) IF on Test.



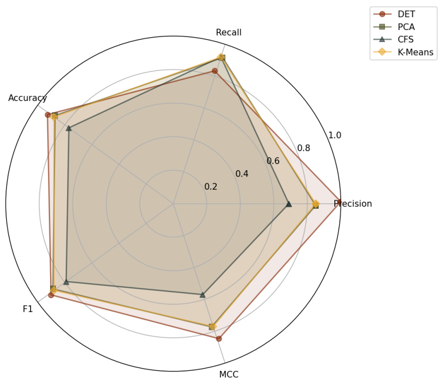
(c) GMM on Val.



(d) GMM on Test.



(e) AE on Val.



(f) AE on Test.

Figure 6.25: Radar plots comparing the performance of *Isolation Forest (IF)*, *Gaussian Mixture Model (GMM)*, and *Autoencoder (AE)* on the Validation (Val) and Test sets generated for TEST 9. Each plot illustrates the metrics obtained using feature subsets derived from four Feature Selection Methods: *Domain Expert Technique (DET)*, *Principal Component Analysis (PCA)*, *Correlation-Based Feature Selection (CFS)*, and *K-Means Clustering-Based Feature Selection (K-Means)*.

Chapter 7

Conclusions

The objective of this thesis was to design, implement, and evaluate an anomaly detection system based on Artificial Intelligence techniques applied to a railway environment, with focus on the monitoring of Wi-Fi radio communication networks supporting railway signalling systems. In this context, the ability to identify anomalous conditions represents a crucial requirement to ensure both the safety of railway operations and the operational and design efficiency of the infrastructure.

One of the main challenges encountered during this work was the initial lack of field-collected data characterized by a low number of anomalies. Indeed, the developed tool was conceived to support communication engineers during the Wi-Fi network design phase; however, effective training of anomaly detection models requires datasets predominantly representative of normal system behaviour. This limitation motivated one central research question of this thesis: whether it is possible to design and integrate synthetic data generation techniques capable of producing high-quality data, preserving the statistical and operational characteristics of real measurements, while enabling reliable training and evaluation of anomaly detection models.

The evaluation of different Synthetic Data Generation (SDG) methods (section 6.1) highlighted substantial differences in their ability to reproduce the statistical, temporal, and structural properties of real data. Across distance-based metrics, latent space analyses, and feature distribution inspections, Forest-Diffusion and TVAE consistently emerged as the most effective approaches, achieving a favorable balance between similarity to real data and sufficient variability. Conversely, more complex generative models, including transformer-based approaches (Chronos and TimesFM), did not outperform simpler methods and often generated synthetic data that were easily distinguishable from real samples. The inclusion of SMT-based constraints ensured logical consistency between features but generally degraded statistical fidelity. Overall, the results indicate that, in the considered scenario, TVAE can be selected as the most suitable model for the data generation pipeline, as it consistently achieved the best performance in the Discriminator Two-Sample Test. In particular, the pipeline composed by TVAE as SDG method and a rejection sampling technique based on fooling the Random Forest classifier, has been chosen has the most suitable approach for this scenario.

The study on contamination factor estimation (section 6.2) highlighted the critical impact of this hyperparameter on the performance of the Isolation Forest model. Among the evaluated strategies, the Zero Anomalies Assumption (ZAA) consistently achieved the most robust results across all experimental settings, yielding the highest Accuracy, F1-score, and Matthews Correlation Coefficient in the majority of tests. This indicates that, in the considered scenario, assuming a contamination factor equal to zero allows the model to better capture anomalies in data while limiting false detections. Automatic estimation methods, such as AE and γGMM, exhibited perfect recall but suffered from a higher number of false anomalies. Domain-based heuristic approaches (DBH-S and DBH-T) proved overly conservative, achieving low false positive rates at the expense of a significantly higher false negative rate.

Overall, the results of the thresholding methods study (section 6.3) revealed that, among the strategies analyzed, the Standard Deviation Inflection

Point (SDIP) method emerges as the most robust and reliable approach, achieving high performance across all datasets and for both reconstruction signals (MAE and SSE). The Z-Score method also demonstrates competitive results. In contrast, clustering-based methods (K-Means and DBS) exhibit variable performance. While DBSCAN systematically underperforms.

Moreover, the experiments on varying the ratio between real and synthetic data (section 6.4) highlighted the critical importance of including real samples in the training of anomaly detection models. Across all backbones (RED, BLUE, and GREEN), the Isolation Forest (IF), Gaussian Mixture Models (GMM), and Autoencoder (AE) exhibit substantially degraded performance when trained exclusively on synthetic data, confirming that synthetic samples alone are insufficient to capture the complexity of real-world anomalies. Increasing the proportion of real data in the training set consistently improves model performance, with a stabilization observed once real data constitute approximately 50% of the training samples. Among the three models, the Autoencoder demonstrates the most stable behavior, showing a clear monotonic improvement as the proportion of real data increases. In contrast, the Isolation Forest and GMM occasionally exhibit oscillatory behavior. These results indicate that, for hybrid datasets combining real and synthetic samples, a balanced composition, where real data account for at least half of the training set, represents an effective compromise, ensuring reliable generalization.

The anomaly detection results (section 6.5) across the nine conducted tests indicate that the Autoencoder (AE) consistently outperforms the other models, achieving the highest overall performance in five of the nine tests. In particular, AE demonstrates superior anomaly detection capabilities across both validation and test datasets, providing robust and reliable identification of anomalies with minimal false negatives. The Gaussian Mixture Model (GMM) obtained the best results in three tests, excelling especially in minimizing the False Negative Rate (FNR). Conversely, the Isolation Forest (IF) model proved to be the least effective overall, outperforming the other models in only a single test and exhibiting high variability, particularly in terms of FNR. These findings suggest that, in this scenario, the Autoencoder represents the most reliable and effective model.

The analysis of the feature selection results (section 6.6) across all nine tests reveals that Principal Component Analysis (PCA) and K-Means clustering-based selection emerge as the most effective strategies, particularly when paired with the Gaussian Mixture Model (GMM) and the Autoencoder (AE), achieving high F1 Scores and low False Negative Rates. In contrast, the Domain Expert Technique (DET) and Correlation-Based Feature Selection (CFS) exhibit greater variability. Across all models, Isolation Forest (IF) is the most sensitive to feature selection, frequently underperforming regardless of the method, while AE demonstrates the strongest robustness and generalization. However, it is important to note that, in this scenario, the application of feature selection methods generally leads to suboptimal results or performances comparable to those obtained using the full set of features, so no clear advantage is observed in their adoption. This is particularly relevant considering that the anomaly detection models considered are relatively small in terms of parameters, meaning that feature selection does not provide a significant improvement in inference time for potential online applications. Thus, the feature selection methods practical benefit over using the complete feature set is limited.

Overall, the conducted study could have concrete implications for the railway domain, particularly in terms of operational safety and efficiency. Thanks to this work, a Radio Network Anomaly Detection tool based on Artificial Intelligence technologies has been developed to support the design of Wi-Fi networks. The tool is designed to ensure continuity and stability of the connection between subsystems, which are essential for the correct operation of railway signaling architectures and for guaranteeing the safe movement of trains. The integrated anomaly detection techniques allow for the rapid identification of interference, insufficient coverage, or congestion, avoiding prolonged manual analyses and reducing design times. Furthermore, the application of the tool optimizes resources by reducing rework costs and post-installation corrective interventions, while supporting the generation of clear and traceable reports, ensuring alignment of network performance with system requirements. Thus, this work could represents a significant contribution to the design of complex radio networks, trying to provide evidence of how the integration of Artificial Intelligence techniques can enhance the efficiency, reliability, and safety of railway networks.

Considering the overall work carried out, two main limitations can be identified. First, the lack of standardization in data collection and sampling periods prevented the adoption of anomaly detection models capable of capturing temporal dependencies within the data. As a consequence, anomaly patterns that are purely temporal in nature cannot be effectively identified by the approaches employed in this study. Second, the proposed synthetic data generation method, based on a Tabular Variational Autoencoder (TVAE) combined with a rejection sampling process, relies on a single classifier (Random Forest) to discriminate between generated and original data. As a result, the effectiveness of this method is strongly dependent on the limitations and biases of the chosen classifier.

Despite the encouraging results obtained in this work, several aspects warrant further investigation and development. First, data collection could be standardized in order to enable the evaluation of anomaly detection models capable of capturing temporal dependencies, or alternatively, to address this limitation by aligning the temporal windows of such models with the sampling period used during data acquisition. Moreover, additional anomaly detection approaches, including both temporal and non-temporal methods based on different paradigms (such as Local Outlier Factor [6], k-Nearest Neighbors, and Support Vector Machines [60]) could be explored. Finally, future work may investigate alternative synthetic data generation techniques, including

GAN-based architectures designed to preserve temporal dependencies (e.g., TimeGAN [67]) and transformer-based models with few-shot prompting for both temporal and tabular data.

Appendix A

Percentile Thresholding Results

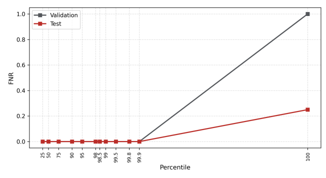
This appendix presents the results obtained from the evaluation described in subsection 5.2.2, applied to the percentile-based thresholding method. Different percentile values were tested across the datasets derived from the three backbones.

As shown in Figure A.1 and Figure A.2, for the RED backbone, setting the threshold at the 99.9th percentile appears to be the optimal choice. This configuration achieves the highest values of *Accuracy*, *F1 Score*, and *MCC*, while maintaining a *False Negative Rate (FNR)* equal to zero.

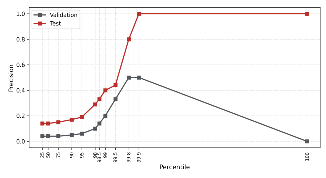
In contrast, for the BLUE backbone, imposing such a high percentile does not yield good results. As evident from both the MAE (Figure A.3) and SSE (Figure A.4) reconstruction errors, performance,measured in terms of *Accuracy*, *F1 Score*, and *MCC*,starts to degrade beyond the 95th percentile. This suggests that overly strict thresholds may lead to an excessive number of false negatives.

For the GREEN backbone, a suitable trade-off is observed around the 99.5th percentile, which produces the best results in terms of *Accuracy*, *F1 Score*, and *MCC*, while keeping the *FNR* reasonably low, as shown in Figure A.5 and Figure A.6.

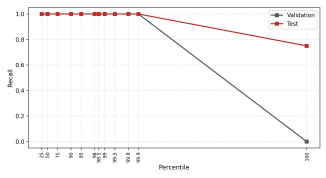
However, it is important to note that the optimal percentile value varies significantly across backbones. This variability highlights one of the main limitations of percentile-based thresholding: its strong dependence on the underlying data distribution. Because the percentile threshold must be empirically tuned for each dataset, the method lacks generalization and reproducibility. Consequently, percentile-based thresholding cannot be reliably applied in this unsupervised setting, as it requires prior knowledge or a great amount of labeled validation data to determine the appropriate threshold for each backbone.



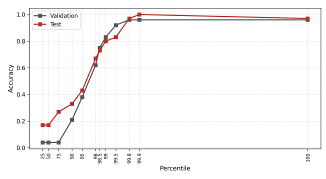
(a) FNR



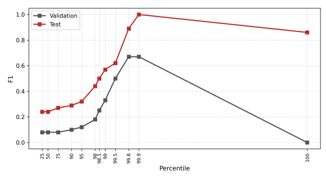
(b) Precision



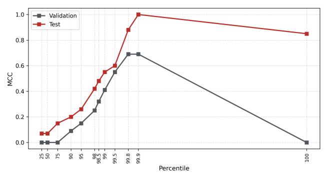
(c) Recall



(d) Accuracy

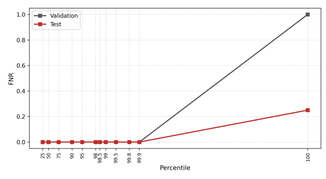


(e) F1 Score

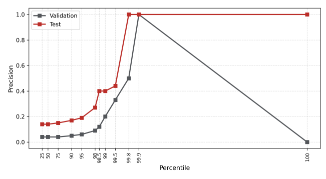


(f) MCC

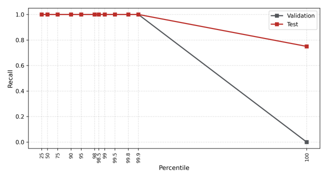
Figure A.1: Percentile-based thresholding performance on the RED backbone using the Mean Absolute Error (MAE) reconstruction error. Each plot shows validation and test trends across increasing percentile thresholds.



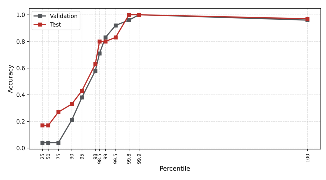
(a) FNR



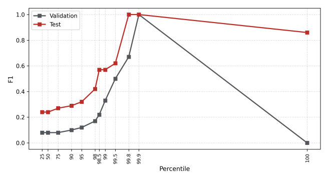
(b) Precision



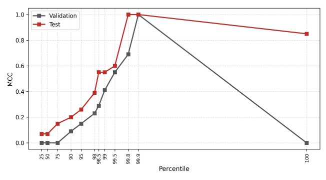
(c) Recall



(d) Accuracy

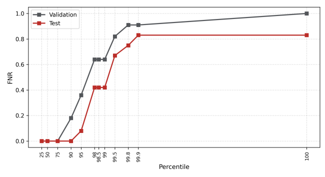


(e) F1 Score

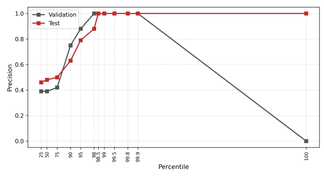


(f) MCC

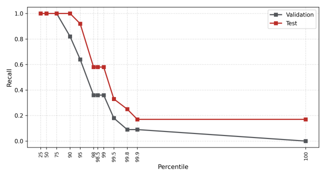
Figure A.2: Percentile-based thresholding performance on the RED backbone using the Sum of Squared Errors (SSE) reconstruction error. Each plot shows validation and test trends across increasing percentile thresholds.



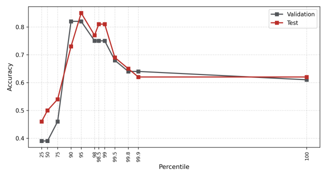
(a) FNR



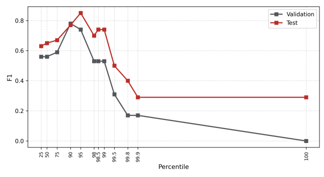
(b) Precision



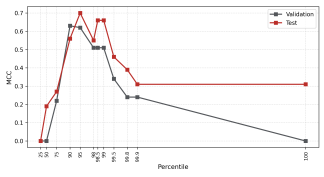
(c) Recall



(d) Accuracy

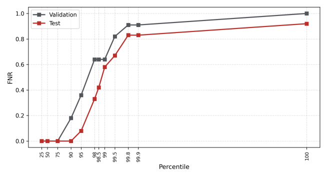


(e) F1 Score

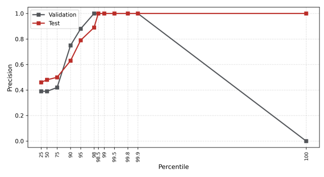


(f) MCC

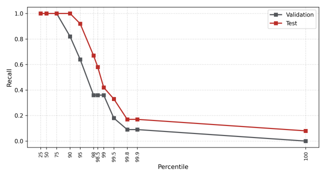
Figure A.3: Percentile-based thresholding performance on the BLUE backbone using the Mean Absolute Error (MAE) reconstruction error. Each plot shows validation and test trends across increasing percentile thresholds.



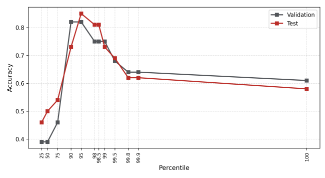
(a) FNR



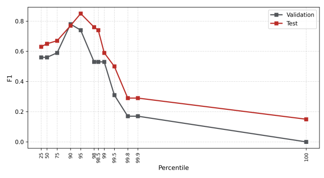
(b) Precision



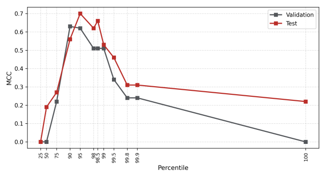
(c) Recall



(d) Accuracy

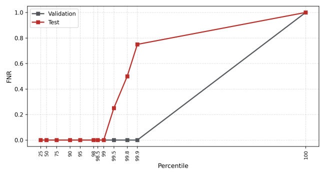


(e) F1 Score

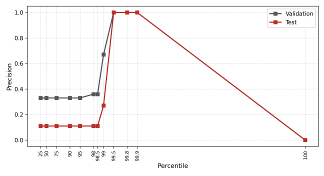


(f) MCC

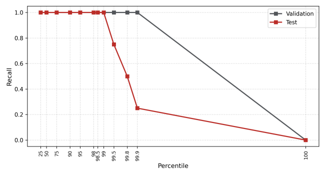
Figure A.4: Percentile-based thresholding performance on the BLUE backbone using the Sum of Squared Errors (SSE) reconstruction error. Each plot shows validation and test trends across increasing percentile thresholds.



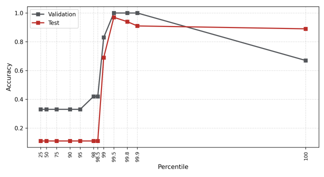
(a) FNR



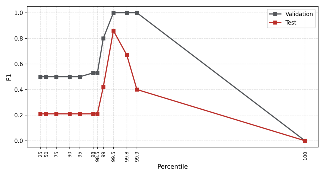
(b) Precision



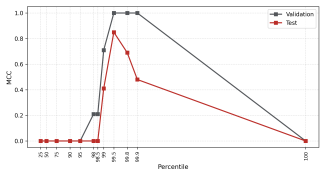
(c) Recall



(d) Accuracy

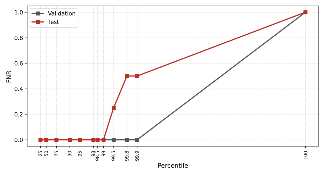


(e) F1 Score

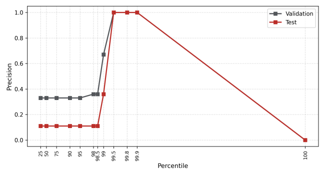


(f) MCC

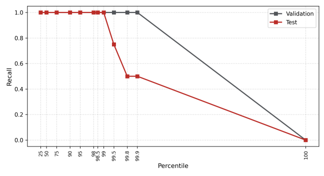
Figure A.5: Percentile-based thresholding performance on the GREEN backbone using the Mean Absolute Error (MAE) reconstruction error. Each plot shows validation and test trends across increasing percentile thresholds.



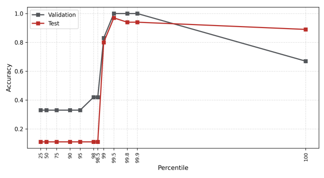
(a) FNR



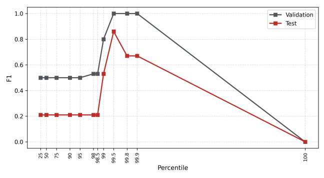
(b) Precision



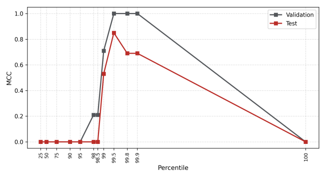
(c) Recall



(d) Accuracy



(e) F1 Score



(f) MCC

Figure A.6: Percentile-based thresholding performance on the GREEN backbone using the Sum of Squared Errors (SSE) reconstruction error. Each plot shows validation and test trends across increasing percentile thresholds.

Appendix B

Results Feature Selection

This appendix reports the complete lists of features selected by each feature selection method, namely *Correlation-based Feature Selection (CFS)*, *Principal Component Analysis (PCA)*, and *K-Means-based selection*, across the three datasets corresponding to the *RED*, *BLUE*, and *GREEN* backbones. As discussed in subsection 5.2.5, each method was applied independently to the datasets, leading to potentially different subsets of relevant features depending on the underlying data structure and statistical assumptions of each approach. The following tables summarize the selected features for each combination of dataset and selection method.

Table B.1: Selected features for the RED backbone.

Selection Method	**Selected Features**
CFS	Bytes Rcvd by E2 (Streaming), Bytes Sent by E1, Consecutive Lost Datagrams: 4–5, Jitter (delay variation): 31–50 (ms), Jitter (delay variation): 51+ (ms), Jitter (delay variation) Maximum (ms), One-Way Delay (ms), RFC 1889 Jitter (ms), RSSI, Throughput (Mbps)
PCA	Bytes Rcvd by E2 (Streaming), Consecutive Lost Datagrams: 1, Consecutive Lost Datagrams: 4–5, Consecutive Lost Datagrams: 6–10, Jitter (delay variation): 11–20 (ms), Jitter (delay variation): 21–30 (ms), Jitter (delay variation): 31–50 (ms), Jitter (delay variation): 51+ (ms), Jitter (delay variation) Maximum (ms), RSSI, Throughput (Mbps)
K-Means	Bytes Lost from E1 to E2, Bytes Rcvd by E2 (Streaming), Bytes Sent by E1, Consecutive Lost Datagrams: 1, Consecutive Lost Datagrams: 2–3, Consecutive Lost Datagrams: 4–5, Consecutive Lost Datagrams: 6–10, Consecutive Lost Datagrams: 11+, Jitter (delay variation): 0–10 (ms), Jitter (delay variation): 11–20 (ms), Jitter (delay variation): 21–30 (ms), Jitter (delay variation): 31–50 (ms), Jitter (delay variation): 51+ (ms), Jitter (delay variation) Maximum (ms), Maximum Consecutive Lost Datagrams, One-Way Delay (ms), RFC 1889 Jitter (ms), RSSI, Throughput (Mbps)

Table B.2: Selected features for the BLUE backbone.

Selection Method	**Selected Features**
CFS	Bytes Rcvd by E2 (Streaming), Bytes Sent by E1, Jitter (delay variation): 0–10 (ms), Jitter (delay variation): 21–30 (ms), Jitter (delay variation): 31–50 (ms), One-Way Delay (ms), RFC 1889 Jitter (ms), RSSI, Throughput (Mbps)
PCA	Bytes Rcvd by E2 (Streaming), Consecutive Lost Datagrams: 1, Consecutive Lost Datagrams: 2–3, Consecutive Lost Datagrams: 6–10, Jitter (delay variation): 11–20 (ms), Jitter (delay variation): 21–30 (ms), Jitter (delay variation): 31–50 (ms), Jitter (delay variation): 51+ (ms), Jitter (delay variation) Maximum (ms), One-Way Delay (ms), RFC 1889 Jitter (ms), Throughput (Mbps)
K-Means	Bytes Lost from E1 to E2, Bytes Rcvd by E2 (Streaming), Bytes Sent by E1, Consecutive Lost Datagrams: 1, Consecutive Lost Datagrams: 2–3, Consecutive Lost Datagrams: 4–5, Consecutive Lost Datagrams: 6–10, Consecutive Lost Datagrams: 11+, Jitter (delay variation): 0–10 (ms), Jitter (delay variation): 11–20 (ms), Jitter (delay variation): 21–30 (ms), Jitter (delay variation): 31–50 (ms), Jitter (delay variation): 51+ (ms), Jitter (delay variation) Maximum (ms), Maximum Consecutive Lost Datagrams, One-Way Delay (ms), RFC 1889 Jitter (ms), RSSI, Throughput (Mbps)

Table B.3: Selected features for the GREEN backbone.

Selection Method	**Selected Features**
CFS	Bytes Rcvd by E2 (Streaming), Bytes Sent by E1, One-Way Delay (ms), RFC 1889 Jitter (ms), RSSI, Throughput (Mbps)
PCA	Bytes Rcvd by E2 (Streaming), Consecutive Lost Datagrams: 1, Consecutive Lost Datagrams: 2–3, Consecutive Lost Datagrams: 4–5, Consecutive Lost Datagrams: 6–10, Consecutive Lost Datagrams: 11+, Jitter (delay variation): 11–20 (ms), Jitter (delay variation): 21–30 (ms), Jitter (delay variation): 31–50 (ms), Jitter (delay variation): 51+ (ms), Jitter (delay variation) Maximum (ms), Throughput (Mbps)
K-Means	Bytes Lost from E1 to E2, Bytes Rcvd by E2 (Streaming), Bytes Sent by E1, Consecutive Lost Datagrams: 1, Consecutive Lost Datagrams: 2–3, Consecutive Lost Datagrams: 4–5, Consecutive Lost Datagrams: 6–10, Consecutive Lost Datagrams: 11+, Jitter (delay variation): 0–10 (ms), Jitter (delay variation): 11–20 (ms), Jitter (delay variation): 21–30 (ms), Jitter (delay variation): 31–50 (ms), Jitter (delay variation): 51+ (ms), Jitter (delay variation) Maximum (ms), Maximum Consecutive Lost Datagrams, One-Way Delay (ms), RFC 1889 Jitter (ms), RSSI, Throughput (Mbps)

Appendix C

Implementation as Software Component

Considering the complexity of the application scenario addressed in this work, it is essential, in an industrial context, to rely on software tools that are both intuitive and easy to use. Moreover, good engineering practice requires code to be structured in a clear and readable manner, so as to facilitate future maintenance and extensibility. For these reasons, two dedicated software tools were developed, each equipped with a simple graphical user interface. The purpose of this appendix is to briefly present these tools, outlining their functionality and overall structure.

C.1 Data Builder Software

Since the field data were collected through two separate software tools, as described in section 4.1, it was necessary to design a dedicated software application capable of ingesting these data and constructing the datasets used throughout this thesis. Specifically, the collected data were stored in two distinct files generated by the two scripts: the first contained the Received Signal Strength Indicator (RSSI) values measured by the Onboard Radio Modem (OBM) during the train journey with respect to each Radio Access Point along the railway line, while the second file included the remaining features described in detail in chapter 4. The software presented in this appendix combines the outputs of these two scripts with an additional technical document mapping MAC addresses to the physical locations of the Radio Access Points along the line, in order to generate the final datasets employed in this work.

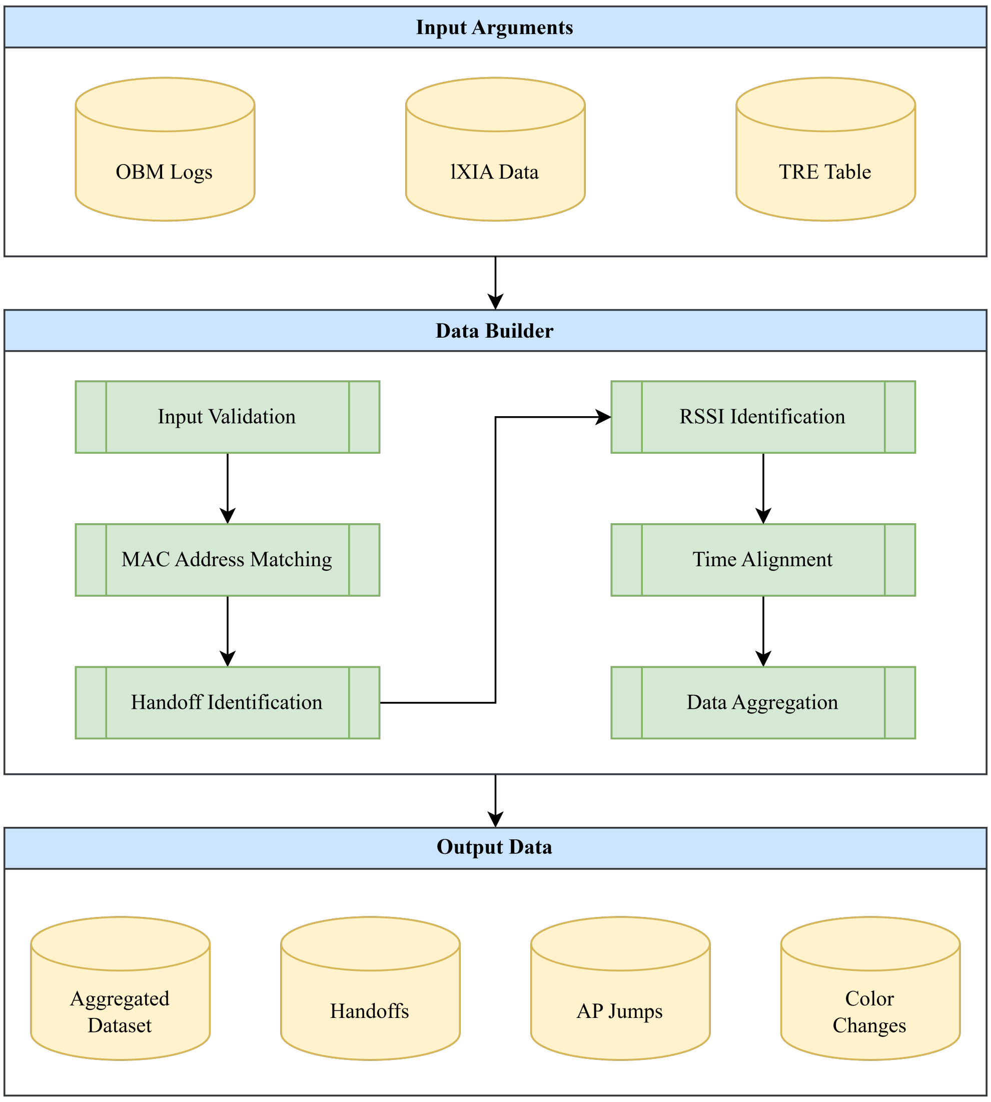


Figure C.1: Main components and workflow of the *Data Builder* software developed in this work. The software is designed to construct the datasets used throughout the study by integrating and interpolating data from multiple sources.

The main modules and operational workflow of this software are summarized in Figure C.1. The application processes three primary data sources.

The first consists of OBM logs, which record the RSSI values measured between the Onboard Radio Modem (OBM) and the Radio Access Points (APs) along the line, as well as the hand-off events occurring during the train journey. The second source is represented by the data collected through the *lXIA* script, which provides real-time measurements of communication quality along the line. The third data source is a technical reference table mapping TRE-BOX identifiers (and, consequently, the associated APs) to their corresponding MAC addresses, which is used to approximate the train position along the route.

The software first verifies the consistency and chronological compatibility of the input resources. Subsequently, MAC addresses are extracted from the textual OBM logs and validated against the entries reported in the TRE table. Hand-off events are then identified in order to determine the specific AP to which the OBM was connected at each time instant. Based on this information, the program selects only the RSSI values associated with the AP actively serving the OBM at each timestamp. Since the OBM logs and the lXIA data streams are not temporally synchronized, a time-alignment procedure is applied according to the methodology described in section 4.1. The aligned data are finally aggregated to construct the final dataset (*Aggregated Dataset*).

In addition to the aggregated dataset, the software produces several auxiliary outputs, including a complete record of hand-off events observed during the train journey (*Handoffs*), a list of “jumps”, i.e., APs to which the OBM did not connect despite being expected to do so (*AP Jumps*), and all instances in which the OBM switched between the RED and BLUE backbones (*Color Changes*).

To facilitate usability, a simple graphical user interface was developed using the Python *Tkinter* library. An example of this interface is shown in Figure C.2. The user can specify the required input files through dedicated fields on the left-hand side of the UI, using the “Browse” button to access the local file system. The “Clear” and “Clear All” buttons allow the input fields to be reset. Upon clicking the "Submit" button, the program is executed and its run-time output is displayed in the textual panel located on the right-hand side of the interface.

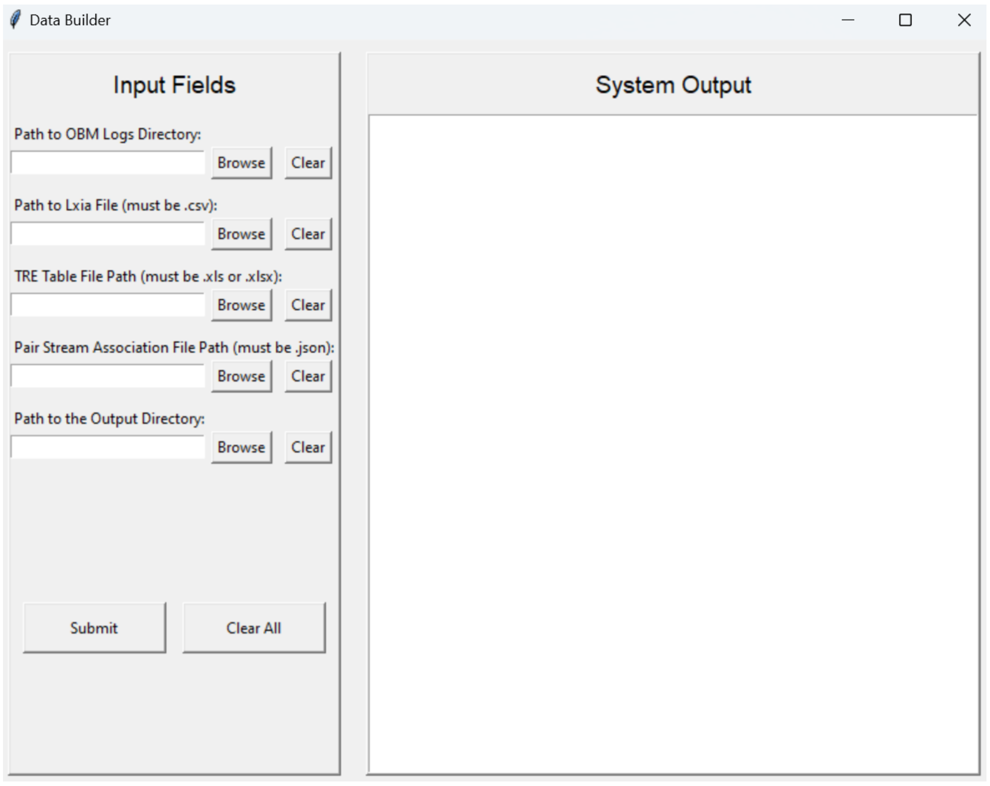

Data Builder
Input Fields
Path to OBM Logs Directory:
Browse
Clear
Path to Lxia File (must be .csv):
TRE Table File Path (must be .xls or .xlsx):
Pair Stream Association File Path (must be .json):
Path to the Output Directory:
Submit
Clear All
System Output

Figure C.2: Graphical User Interface of the *Data Builder* software developed in this work. The input parameters can be specified through the fields on the left side of the interface, while the program output is displayed in the text panel on the right, providing the user with a clear and continuous feedback on the execution process.

C.2 Anomaly Detector Software

In order to facilitate the identification of anomalies, a software tool was designed to support network designers during the development phase. The purpose of this software, presented in this appendix, is to process the datasets generated by the *Data Builder* program and produce human-friendly output.

The main functionality and modules of this software are summarized in the diagram shown in Figure C.3. The tool first verifies that the input data are

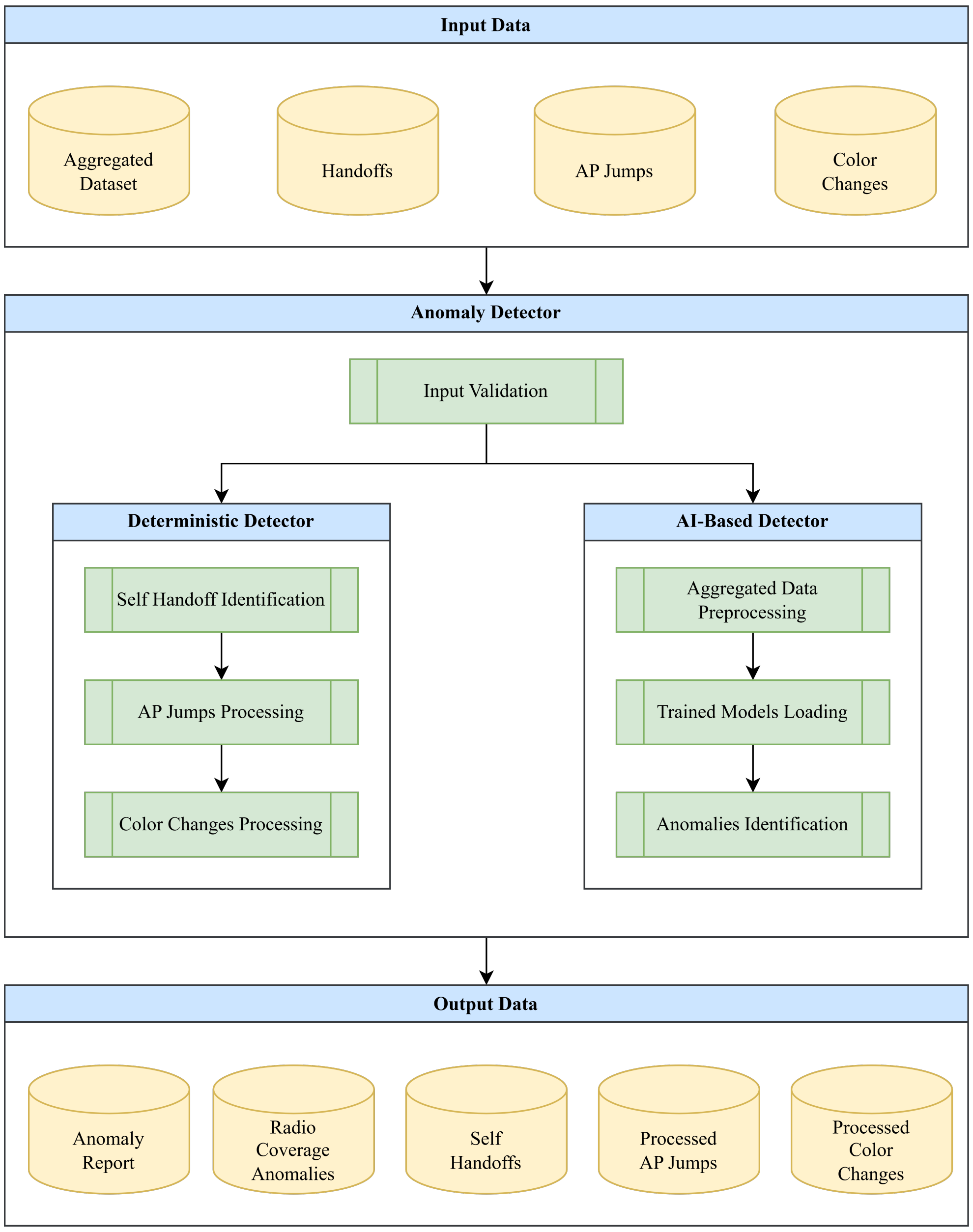


Figure C.3: Main components and workflow of the *Anomaly Detector* software developed in this work. The software is designed to identify anomalies in the datasets generated using the *Data Builder* program. It produces a report summarizing the detected anomalies, as well as additional output files containing detailed information about the identified issues.

properly structured and temporally consistent, in accordance with the specifications presented in chapter 4. Since the types of anomalies in the dataset can be broadly categorized as either deterministic or related to radio signals, the software instantiates two distinct detectors: the *Deterministic Detector* and the *AI-Based Detector*. The Deterministic Detector identifies all self-handoff events by analyzing the historical hand-off logs. It then processes AP Jumps and Color Changes, selecting only those events that are temporally consistent with the input data, producing a user-friendly representation of these events. The AI-Based Detector, instead, preprocesses the dataset generated by the *Data Builder*, applying feature selection and normalization to prepare the data for AI models. It subsequently loads the pre-trained models (Isolation Forest, Gaussian Mixture Model, and Autoencoder) for each backbone (RED, BLUE, and GREEN), which are then used to detect anomalies and associate them with the corresponding APs using their MAC addresses.

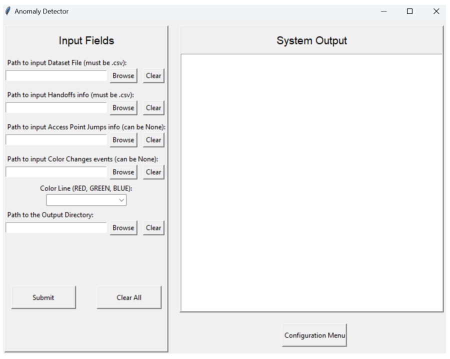
The software produces an output document in Portable Document Format called the *Anomaly Report*, summarizing both deterministic and AI-detected anomalies. In addition, several supplementary output files are generated to provide more detailed information on the anomalies identified. Specifically, the tool produces files representing all deterministic anomaly events, as well as *Radio Coverage Anomalies*, which contain all anomalies detected by at least one AI model. Each anomaly is annotated with the corresponding timestamp, the MAC address of the AP to which the OBM was connected, and the models that detected it.

Furthermore, an *Anomaly Score* is calculated for each detected anomaly, representing the confidence of the tool in identifying that anomaly. This score is computed as a weighted average of the predictions from the models, allowing each anomaly to be associated with a percentage-based confidence value. The weights can be specified by the user via a configuration file and are calculated based on model performance during testing using the following heuristic:

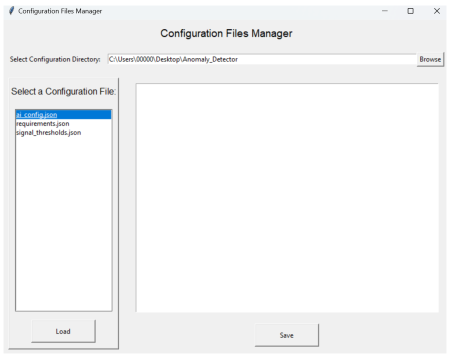
$$w_i = \frac{\frac{1}{K}\sum_{k=1}^{K}\frac{1}{\text{FNR}_{i,k}}}{\sum_{j=1}^{n}\left(\frac{1}{K}\sum_{k=1}^{K}\frac{1}{\text{FNR}_{j,k}}\right)}$$

- w_i: Weight assigned to model i.
- $\text{FNR}_{i,k}$: False Negative Rate of model i in test k.
- K: Total number of tests.
- n: Total number of models.

This weighting scheme penalizes predictions from models that performed poorly during testing by assigning them a lower weight, based on their observed False Negative Rate.


Anomaly Detector
Input Fields
Path to input Dataset File (must be .csv):
Browse
Clear
Path to input Handoffs info (must be .csv):
Browse
Clear
Path to input Access Point Jumps info (can be None):
Browse
Clear
Path to input Color Changes events (can be None):
Browse
Clear
Color Line (RED, GREEN, BLUE):
Path to the Output Directory:
Browse
Clear
Submit
Clear All
System Output
Configuration Menu

(a) User interface of the *Anomaly Detector*. Input parameters can be specified through the fields on the left side of the interface, while the program output is displayed in the text panel on the right, providing the user with clear and continuous feedback on the execution process.


Configuration Files Manager
Configuration Files Manager
Browse
Select a Configuration File:
Load
Save

Select Configuration Directory: C:\Users\00000\Desktop\Anomaly_Detector

ai_config.json
requirements.json
signal_thresholds.json

(b) User interface of the *Configuration File Manager*. The window on the left allows the user to select a configuration file, which can then be modified using the interface on the right.

Figure C.4: Graphical user interfaces of the software developed in this work, including both the *Anomaly Detector* (left) and the *Configuration File Manager* (right). The interfaces are designed to facilitate intuitive interaction, allowing the user to input parameters, monitor execution, and manage configuration files efficiently.

To facilitate the use of this tool, a graphical user interface was implemented, as shown in Figure C.4. Through this interface, the user can easily provide input parameters using the “Browse”, “Clear”, and “Clear All” buttons.

By pressing the “Submit” button, the program executes using the specified parameters and the configuration files stored in the tool’s working directory. These configuration files can also be viewed and modified directly within the tool via the *Configuration File Manager*, which can be accessed by clicking the “Configuration Menu” button.

Appendix D

Experimental Setup

The purpose of this appendix is to describe the implementation details of the proposed work. In particular, it outlines the computational environment adopted for training and testing the models, the software libraries employed together with their respective versions, and the hyperparameter selection strategy used for the experimental evaluation.

All experiments were implemented in Python [47], version 3.12.0, using Jupyter Notebook [46] as the development environment.

The training and evaluation of the synthetic data generation models were performed on Google Colab [16], leveraging a Tesla T4 GPU. This architecture is composed of 40 Streaming Multiprocessors (SMs) with a shared 6 MB L2 cache, and is equipped with 16 GB of high-bandwidth GDDR6 memory directly connected to the processor.

Conversely, the training and testing of the anomaly detection models were conducted on a standard laptop without GPU acceleration. Specifically, all experiments were executed on a 13th Gen Intel(R) Core(TM) i5-1335U CPU (1.30 GHz), equipped with 8.00 GB of RAM. This choice reflects the practical deployment scenario of the proposed tool, which is intended to operate on commonly available hardware.

The main software libraries used in this work, together with their corresponding versions, are summarized in Table D.1.

Library	Version	Library	Version
numpy[38]	2.1.3	sdv[57]	1.21.0
pandas[40]	2.2.3	gretel-synthetics[17]	0.22.19
regex[48]	2.5.148	pysmt[13]	0.9.6
xlrd[64]	2.0.1	z3 solver[68]	4.13.0.0
tabulate[49]	0.9.0	tslearn[58]	0.6.3
joblib[23]	1.5.1	timesfm[9]	1.2.9
reportlab[52]	4.4.2	chronos-forecasting[2]	1.5.2
scikit-learn[53]	1.6.1	ForestDiffusion[25]	1.0.6
tensorflow[59]	2.19.0	optuna[1]	4.4.0

Table D.1: Major software libraries and versions used in this work.

Regarding the anomaly detection models employed in this study, their hyperparameters were selected through a grid search procedure. Specifically, the tuning process included the number of estimators for the Isolation Forest, the number of components, covariance type, and covariance regularization parameters for Gaussian Mixture Models, as well as the number of layers, number of neurons per layer, learning rate, and optimizer choice for the Autoencoder. The final Autoencoder architecture obtained through this process is illustrated in Figure D.1.

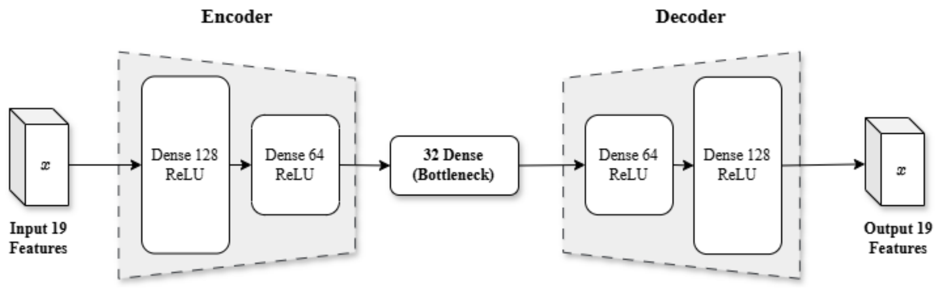


Figure D.1: Graphical representation of the Autoencoder model used in this thesis. The architecture is symmetric, consisting of an encoder with layers of 128, 64, and 32 (latent) neurons, and a decoder mirroring the encoder with layers of 64, 128, and 19 neurons.

Finally, for the synthetic data generation pipeline based on the TVAE+RS approach, the Tabular Variational Autoencoder implemented within the SDV framework was trained for 3000 epochs, while the Random Forest classifier used in the rejection sampling stage was trained using 100 estimators.

Bibliography

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: a next-generation hyperparameter optimization framework. In *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

[2] A. F. Ansari, L. Stella, A. C. Türkmen, X. Zhang, P. Mercado, H. Shen, O. Shchur, S. S. Rangapuram, S. Pineda-Arango, S. Kapoor, J. Zschiegner, D. C. Maddix, H. Wang, M. W. Mahoney, K. Torkkola, A. G. Wilson, M. Bohlke-Schneider, and B. Wang. Chronos: learning the language of time series. *Trans. Mach. Learn. Res.*, 2024, 2024. URL: `https://openreview.net/forum?id=gerNCVqqtR`.

[3] D. Bank, N. Koenigstein, and R. Giryes. Autoencoders. *CoRR*, abs/2003.05991, 2020. arXiv: `2003.05991`. URL: `https://arxiv.org/abs/2003.05991`.

[4] A. Bauer, S. Trapp, M. Stenger, R. Leppich, S. Kounev, M. Leznik, K. Chard, and I. T. Foster. Comprehensive exploration of synthetic data generation: A survey. *CoRR*, abs/2401.02524, 2024. DOI: `10.48550/ARXIV.2401.02524`. arXiv: `2401.02524`. URL: `https://doi.org/10.48550/arXiv.2401.02524`.

[5] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011. arXiv: `1106.1813`. URL: `http://arxiv.org/abs/1106.1813`.

[6] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 93–104. ACM, 2000. DOI: 10.1145/342009.335388. URL: https://doi.org/10.1145/342009.335388.

[7] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 785–794. ACM, 2016. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785.

[8] A. Coletta, S. Gopalakrishnan, D. Borrajo, and S. Vyetrenko. On the constrained time-series generation problem. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/bfb6a69c0d9e2bc596e1cd31f16fcdde-Abstract-Conference.html.

[9] A. Das, W. Kong, R. Sen, and Y. Zhou. A decoder-only foundation model for time-series forecasting. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL: https://openreview.net/forum?id=jn2iTJas6h.

[10] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In E. Simoudis,

J. Han, and U. M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231. AAAI Press, 1996. URL: `http://www.aaai.org/Library/KDD/1996/kdd96-037.php`.

[11] Y. Fan, L. Zhang, and K. Li. Ae-bilstm: multivariate time-series EMI anomaly detection in 5g-r high-speed rail wireless communications. In *IEEE International Conference on Communications Workshops, ICC 2024 Workshops, Denver, CO, USA, June 9-13, 2024*, pages 439–444. IEEE, 2024. DOI: `10.1109/ICCWORKSHOPS59551.2024.10615719`. URL: `https://doi.org/10.1109/ICCWorkshops59551.2024.10615719`.

[12] G. Forestier, F. Petitjean, H. A. Dau, G. I. Webb, and E. J. Keogh. Generating synthetic time series to augment sparse datasets. In V. Raghavan, S. Aluru, G. Karypis, L. Miele, and X. Wu, editors, *2017 IEEE International Conference on Data Mining, ICDM 2017, New Orleans, LA, USA, November 18-21, 2017*, pages 865–870. IEEE Computer Society, 2017. DOI: `10.1109/ICDM.2017.106`. URL: `https://doi.org/10.1109/ICDM.2017.106`.

[13] M. Gario and A. Micheli. Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT Workshop 2015*, 2015.

[14] C. D. Gobbo. A comparative study of open-source libraries for synthetic tabular data generation: SDV vs. synthcity. *CoRR*, abs/2506.17847, 2025. DOI: `10.48550/ARXIV.2506.17847`. arXiv: `2506.17847`. URL: `https://doi.org/10.48550/arXiv.2506.17847`.

[15] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014. arXiv: `1406.2661`. URL: `http://arxiv.org/abs/1406.2661`.

[16] Google colab. URL: `https://colab.google/` (visited on 01/08/2025).

[17] Gretel synthetics. URL: https://github.com/gretelai/gretel-synthetics (visited on 01/08/2025).

[18] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13:723–773, March 2012.

[19] Z. He, P. Chen, X. Li, Y. Wang, G. Yu, C. Chen, X. Li, and Z. Zheng. A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems. *IEEE Trans. Neural Networks Learn. Syst.*, 34(4):1705–1719, 2023. DOI: 10.1109/TNNLS.2020.3027736. URL: https://doi.org/10.1109/TNNLS.2020.3027736.

[20] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997. DOI: 10.1162/NECO.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[21] Ieee standard for communications-based train control (cbtc) performance and functional requirements. *IEEE Std 1474.1-2025 (Revision of IEEE Std 1474.1-2004)*:1–61, 2025. DOI: 10.1109/IEEESTD.2025.11033725.

[22] U. Islam, R. Q. Malik, A. S. Al-Johani, M. R. Khan, Y. I. Daradkeh, I. Ahmad, K. A. Alissa, Z. Abdul-Samad, and E. M. Tag-Eldin. A novel anomaly detection system on the internet of railways using extended neural networks. *Electronics*, 11(18), 2022. ISSN: 2079-9292. DOI: 10.3390/electronics11182813. URL: https://www.mdpi.com/2079-9292/11/18/2813.

[23] Joblib. URL: https://github.com/joblib/joblib (visited on 01/08/2025).

[24] B. Joel and S. Cole. What is anomaly detection? *IBM Think Blog*, December 2023. URL: https://www.ibm.com/think/topics/anomaly-detection (visited on 09/09/2025).

[25] A. Jolicoeur-Martineau, K. Fatras, and T. Kachman. Generating and imputing tabular data via diffusion and flow-based gradient-boosted trees. In S. Dasgupta, S. Mandt, and Y. Li, editors, *International Conference on Artificial Intelligence and Statistics, 2-4 May 2024, Palau de Congressos, Valencia, Spain*, volume 238 of *Proceedings of Machine Learning Research*, pages 1288–1296. PMLR, 2024. URL: https://proceedings.mlr.press/v238/jolicoeur-martineau24a.html.

[26] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL: http://arxiv.org/abs/1312.6114.

[27] A. Komadina, M. Martinic, S. Gros, and Z. Mihajlovic. Comparing threshold selection methods for network anomaly detection. *IEEE Access*, 12:124943–124973, 2024. DOI: 10.1109/ACCESS.2024.3452168. URL: https://doi.org/10.1109/ACCESS.2024.3452168.

[28] C. Lázaro and C. Angulo. Using umap for partially synthetic healthcare tabular data generation and validation. *Sensors*, 24(23), 2024. ISSN: 1424-8220. DOI: 10.3390/s24237843. URL: https://www.mdpi.com/1424-8220/24/23/7843.

[29] Z. Lin, A. Jain, C. Wang, G. Fanti, and V. Sekar. Generating high-fidelity, synthetic time series datasets with doppelganger. *CoRR*, abs/1909.13403, 2019. arXiv: 1909.13403. URL: http://arxiv.org/abs/1909.13403.

[30] F. T. Liu, K. M. Ting, and Z. Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining ICDM 2008, December 15-19, 2008, Pisa, Italy*, pages 413–422. IEEE Computer Society, 2008. DOI: 10.1109/ICDM.2008.17. URL: https://doi.org/10.1109/ICDM.2008.17.

[31] D. Lopez-Paz and M. Oquab. Revisiting classifier two-sample tests. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=SJkXfE5xx.

[32] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *CoRR*, abs/1607.00148, 2016. arXiv: 1607.00148. URL: http://arxiv.org/abs/1607.00148.

[33] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *23rd European Symposium on Artificial Neural Networks, ESANN 2015, Bruges, Belgium, April 22-24, 2015*, 2015. URL: https://www.esann.org/sites/default/files/proceedings/legacy/es2015-56.pdf.

[34] U. Michelucci. An introduction to autoencoders. *CoRR*, abs/2201.03898, 2022. arXiv: 2201.03898. URL: https://arxiv.org/abs/2201.03898.

[35] A. Moitra. *Gaussian mixture models*. In *Algorithmic Aspects of Machine Learning*. Cambridge University Press, Cambridge, 2018, XX–YY. ISBN: 9781316882177. DOI: 10.1017/9781316882177.

[36] D. Monniaux. A survey of satisfiability modulo theory. In V. P. Gerdt, W. Koepf, W. M. Seiler, and E. V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing - 18th International Workshop, CASC 2016, Bucharest, Romania, September 19-23, 2016, Proceedings*, volume 9890 of *Lecture Notes in Computer Science*, pages 401–425. Springer, 2016. DOI: 10.1007/978-3-319-45641-6_26. URL: https://doi.org/10.1007/978-3-319-45641-6%5C_26.

[37] J. Murel and E. Kavlakoglu. What is a confusion matrix? *IBM Think Blog*. URL: `https://www.ibm.com/think/topics/confusion-matrix` (visited on 10/11/2025).

[38] Numpy. URL: `https://numpy.org/` (visited on 01/08/2025).

[39] A. Origlia, S. D. Martino, and E. Battista. Rail anomalies detection: A comparative analysis of three self-supervised models on real data. *Comput. Ind.*, 148:103909, 2023. DOI: `10.1016/J.COMPIND.2023.103909`. URL: `https://doi.org/10.1016/j.compind.2023.103909`.

[40] Pandas. URL: `https://pandas.pydata.org/` (visited on 01/08/2025).

[41] M. Panja, T. Chakraborty, U. Kumar, and A. Hadid. Probabilistic autoregressive neural networks for accurate long-range forecasting. In B. Luo, L. Cheng, Z. Wu, H. Li, and C. Li, editors, *Neural Information Processing - 30th International Conference, ICONIP 2023, Changsha, China, November 20-23, 2023, Proceedings, Part XIII*, volume 1967 of *Communications in Computer and Information Science*, pages 457–477. Springer, 2023. DOI: `10.1007/978-981-99-8178-6\_35`. URL: `https://doi.org/10.1007/978-981-99-8178-6%5C_35`.

[42] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran. Image transformer. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4052–4061. PMLR, 2018. URL: `http://proceedings.mlr.press/v80/parmar18a.html`.

[43] N. Patki, R. Wedge, and K. Veeramachaneni. The synthetic data vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2016, Montreal, QC, Canada, October 17-19, 2016*,

pages 399–410. IEEE, 2016. DOI: 10.1109/DSAA.2016.49. URL: https://doi.org/10.1109/DSAA.2016.49.

[44] L. Perini, P. Bürkner, and A. Klami. Estimating the contamination factor's distribution in unsupervised anomaly detection. A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, 2023. URL: https://proceedings.mlr.press/v202/perini23a.html.

[45] F. Petitjean, A. Ketterlin, and P. Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognit.*, 44(3):678–693, 2011. DOI: 10.1016/J.PATCOG.2010.09.013. URL: https://doi.org/10.1016/j.patcog.2010.09.013.

[46] Project jupyter. URL: https://jupyter.org/ (visited on 01/08/2025).

[47] Python programming language. URL: https://www.python.org/ (visited on 01/08/2025).

[48] Python regular expression documentation. URL: https://docs.python.org/3/library/re.html (visited on 01/08/2025).

[49] Python tabulate. URL: https://pypi.org/project/tabulate/ (visited on 01/08/2025).

[50] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL: https://jmlr.org/papers/v21/20-074.html.

[51] J.-C. Régin. *Global constraints: a survey*. In *Hybrid Optimization: The Ten Years of CPAIOR*. P. van Hentenryck and M. Milano, editors. Springer New York, New York, NY, 2011, pages 63–134. ISBN: 978-1-4419-1644-0. DOI: 10.1007/978-1-4419-1644-0_3. URL: https://doi.org/10.1007/978-1-4419-1644-0_3.

[52] Reportlab. URL: https://github.com/mattjmorrison/ReportLab (visited on 01/08/2025).

[53] Scikit-learn. URL: https://scikit-learn.org/stable/ (visited on 01/08/2025).

[54] P. Senin. Dynamic time warping algorithm review, January 2009.

[55] M. Stenger, R. Leppich, I. T. Foster, S. Kounev, and A. Bauer. Evaluation is key: a survey on evaluation measures for synthetic time series. In volume 11 of number 1, page 66, 2024. DOI: 10.1186/S40537-024-00924-7. URL: https://doi.org/10.1186/s40537-024-00924-7.

[56] C. Sun, A. Myers, C. Vondrick, K. Murphy, and C. Schmid. Videobert: A joint model for video and language representation learning. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 7463–7472. IEEE, 2019. DOI: 10.1109/ICCV.2019.00756. URL: https://doi.org/10.1109/ICCV.2019.00756.

[57] Synthetic data vault. URL: https://docs.sdv.dev/sdv (visited on 01/08/2025).

[58] R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar, and E. Woods. Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, 21(118):1–6, 2020. URL: http://jmlr.org/papers/v21/20-091.html.

[59] Tensorflow. URL: https://www.tensorflow.org/ (visited on 01/08/2025).

[60] V. Vapnik, S. E. Golowich, and A. J. Smola. Support vector method for function approximation, regression estimation and signal processing. In M. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996*, pages 281–287. MIT Press, 1996. URL: http://papers.nips.cc/paper/1187-support-vector-method-for-function-approximation-regression-estimation-and-signal-processing.

[61] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[62] P. Venkataanusha, C. Anuradha, D. Murty, and S. Chebrolu. Detecting outliers in high dimensional data sets using z-score methodology. *International Journal of Innovative Technology and Exploring Engineering*, 9:48–53, November 2019. DOI: 10.35940/ijitee.A3910.119119.

[63] Y. Wang, X. Du, Z. Lu, Q. Duan, and J. Wu. Improved lstm-based time-series anomaly detection in rail transit operation environments. *IEEE Trans. Ind. Informatics*, 18(12):9027–9036, 2022. DOI: 10.1109/TII.2022.3164087. URL: https://doi.org/10.1109/TII.2022.3164087.

[64] Xlrd. URL: https://xlrd.readthedocs.io/ (visited on 01/08/2025).

[65] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni. Modeling tabular data using conditional GAN. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7333–7343, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/254ed7d2de3b23ab10936522dd547b78-Abstract.html.

[66] Y. Xu and J. Liu. High-speed train fault detection with unsupervised causality-based feature extraction methods. *Adv. Eng. Informatics*, 49:101312,

2021. DOI: 10.1016/J.AEI.2021.101312. URL: https://doi.org/10.1016/j.aei.2021.101312.

[67] J. Yoon, D. Jarrett, and M. van der Schaar. Time-series generative adversarial networks. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5509–5519, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/c9efe5f26cd17ba6216bbe2a7d26d490-Abstract.html.

[68] Z3 theorem prover. URL: https://github.com/Z3Prover/z3 (visited on 01/08/2025).

[69] K. A. Zhang, N. Patki, and K. Veeramachaneni. Sequential models in the synthetic data vault. *CoRR*, abs/2207.14406, 2022. DOI: 10.48550/ARXIV.2207.14406. arXiv: 2207.14406. URL: https://doi.org/10.48550/arXiv.2207.14406.

[70] Y. Zuo, F. Thiery, P. Chandran, J. Odelius, and M. Rantatalo. Squat detection of railway switches and crossings using wavelets and isolation forest. *Sensors*, 22(17):6357, 2022. DOI: 10.3390/S22176357. URL: https://doi.org/10.3390/s22176357.

Acknowledgements

I would like to begin by expressing my sincere gratitude to my supervisor, Professor Andrea Borghesi, for his patience, guidance, and constant support. His availability and encouragement made this thesis and the associated research work possible.

I would also like to thank Professor Paolo Torroni, who gave me the opportunity to embark on this thesis journey and supported me in its initial stages.

I extend my thanks to the entire project team of the Turin metro line 1 and to Alstom for accompanying me throughout this experience and for their availability and collaboration.

Finally, I would like to thank my family, my girlfriend, and my friends, who have always been by my side and whose support has been fundamental throughout this journey. Throughout these long years of study, they have always been by my side, helping me face every challenge along the way. The true strength of a person lies in those who stand beside them, and you have all been my strength.