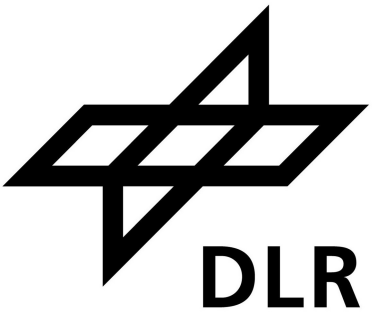
Unsupervised Anomaly Detection for Rocket Engine Test Facilities



BACHELOR THESIS

by

ANSGAR LEMKE

Matriculation Number: 2445192
Degree Program: Aerospace Computer Science
First Supervisor: Prof. Günther Waxenegger-Wilfing
Second Supervisor: Prof. Andreas Nüchter
Advisor: Eldin Kurudzija, M. Sc.

Faculty of Mathematics and Computer Science,
Julius-Maximilians-Universität Würzburg;
Institute of Rocket Propulsion, German Aerospace Center

Würzburg, June 15, 2023

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.
Ich erkläre weiterhin, dass die vorliegende Arbeit in gleicher oder ähnlicher Form noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Würzburg, den 15. Juni 2023 Ansgar Lemke

Abstract

This thesis investigates unsupervised reconstruction-based anomaly detection methods on synthetic multivariate time series data, generated for the Liquid Oxygen system of the P5 rocket engine test facility in Lampoldshausen. To support the health monitoring system based on thresholds for sensor values, the methods should efficiently detect anomalous data. The anomaly detection results could be used on top of the health monitoring system to support the operators when the system suggests a test abort. To achieve these goals, the reconstruction error of a Feedforward Auto-Encoder trained on purely nominal data is used to predict anomalies. The detection rates on the rocket engine test facility data are promising, with an F1 score of 0.90, but improvements on false positives and on more complex fault types are necessary for this anomaly detection approach to be used in actual engine tests.

Kurzfassung

Diese Arbeit befasst sich mit unüberwachten, rekonstruktionsbasierten Ansätzen zur Erkennung von Anomalien auf multivariaten Zeitreihendaten. Diese Daten wurden für das LOX-System des Raketentriebwerkteststandes P5 in Lampoldshausen generiert. Wenn das Health Monitoring System des Teststands zum Testabbruch rät, könnte die Anomaliedetektion bei der Entscheidungsfindung helfen. Dazu wird ein Feedforward-Autoencoder trainiert. Mit Hilfe des Rekonstruktionsfehlers werden dann Anomalien detektiert. Die Detektionsraten bei den Daten für das LOX-System sind mit einem F1-Wert von $0,90$ vielversprechend. Allerdings sind Verbesserungen bei den falsch-positiven Ergebnissen und bei komplexeren Fehlertypen erforderlich, um diesen Ansatz zur Erkennung von Anomalien bei tatsächlichen Triebwerkstests verwenden zu können.

Contents

List of Figures

List of Tables

List of Abbreviations

AD Anomaly Detection

ADAM ADAptive Moment estimation

AE Auto-Encoder

API Application Programming Interface

CPU Central Processing Unit

DLR German Aerospace Center

ESPSS European Space Propulsion System Simulation

EVT Extreme Value Theory

FF Feedforward

FF AE Feedforward Auto-Encoder

FN False Negative

FP False Positive

FPR False Positive Rate

GPU Graphics Processing Unit

HP Hyper-Parameter

LOX Liquid Oxygen

LSTM Long Short-Term Memory

ML Machine Learning

MSE Mean Squared Error

N2 Gaseous Nitrogen

NN Neural Network

PCA Principal Component Analysis

PI Proportional Integral

PL Pytorch Lightning

POT Peaks-Over-Threshold

ROC Receiver Operating Characteristic

TN True Negative

TP True Positive

TPR True Positive Rate

Chapter 1

Introduction

The subject of *Machine Learning* (ML) has been on the rise in recent years. With sufficiently large datasets, ML opens the door to new, unthinkable applications in the respective areas, such as *Large Language Models* like *GPT-3*, image generation with *Stable Diffusion*, or prediction of crime. Even the space sector, with its quite conservative approaches, has not been spared ([1]–[3]), but not without difficulties. Test resources and runtime data in the space sector are constrained by several reasons, such as financial constraints, the interests of state defense, and safety requirements in space that disallow the use of state-of-the-art components. As a result, various systems in this sector cannot provide sufficiently large enough datasets as required by ML.

In the operation of test bench systems for rocket engines, it is crucial to detect anomalies to prevent major destruction in the event of failure. In aerospace, very conservative approaches for a health monitoring system are still widespread. Often enough, anomalies are detected by applying fixed limits to selected sensor values. This approach is known as the *Redlines Method*. If a value exceeds the limit and propagates into potentially dangerous regions, the operation of the test bench is interrupted. The abort can be necessary to prevent mechanical failures, or catastrophic events caused by them. Depending on the application, this creates a significant financial loss. A system that aids the test bench operators with a second opinion on the state of the system can prevent unnecessary test aborts and thus reduce the development costs for new rocket engines.

The LOX system of the test bench *P5* at Lampoldshausen, operated by *German Aerospace Center* (DLR), has been selected to evaluate the potential of ML based Anomaly Detection (AD). This system had been modeled prior to this thesis by Dresia *et al.* [3] using the software *EcosimPro*. By space industry standards, a large amount of operational data is available for this test bench, which could be used for a later fine-tuning of the AD model.

1.1 State of the Art

Several existing methods are used and modified to adapt to the requirements of system and data. This section gives an overview of the topic of AD as well as related work.

1.1.1 Anomaly Detection

This thesis is located in the field of *Anomaly Detection* (AD). AD can be defined as the problem of separating or detecting anomalous patterns from nominal behavior [4]. In the following, these patterns are called anomalies or faults. AD is applied in many different applications such as health care [5], cyber-security [6], fraud detection for credit cards [7], insurance [8], military surveillance [9], as well as detecting faults in components of spacecrafts [1].

Early studies on the subject of AD have started in the 19th century by Edgeworth [10]. Since then, various techniques have been developed for AD, some domain-specific, others more general [4]. The techniques can be categorized by the degree of *supervision*. *Supervised* classification means that all data is labeled for anomalies. In *semi-supervised* methods, parts of the data are labeled. For *unsupervised* AD, labels are missing, and the method has to figure out by itself which values could be anomalous. Another classification of the techniques can be done by sorting them into *reconstruction-based* and *prediction-based* techniques. *Reconstruction-based* techniques aim to transform the data into some *encoded* form from the input and then reconstruct the input data from this *encoded* form. The reconstruction error is then fed into an anomaly classification model. In contrast, *prediction-based* techniques try to predict the values of selected variables which are then compared to the real values in order to calculate a *residual* error. This *residual* error is processed similar to the *reconstruction-based* approach.

In recent years, *Neural Network*s (NNs) have become more important in AD. NNs are capable of learning complex patterns and generalize well to unseen data [11]. They require large amounts of available data to be trained. Thus, any AD application providing enough data can be a target for the implementation of ML-based AD. Often, the data used for AD is *imbalanced* between anomalous and nominal data. AD methods have to take this problem into account and work even on highly *imbalanced* datasets.

1.1.2 Related Work

There exist several relevant works in unsupervised *Anomaly Detection* on time series data. Hundman *et al.* [1] use *Long Short-Term Memory* (LSTM) *Auto-Encoder*s (AEs) on expert-labeled telemetry data from spacecrafts in combination with a dynamic thresholding method and additional techniques to mitigate false-positives. A similar approach is used by ElDali and Kumar [12] who discuss the use of technique based an variable sequence LSTMs model and growing neural networks in order to estimate a *health index* for aircraft engines and satellite attitude actuators. The *health index* is then

used as a fault indicator. The requirements set by the use of aircraft and spacecraft data are similar to those of this thesis. LSTM NNs require a large amount of resources for training. In addition, the proposed thresholding method assumes a long time series, resulting in questionable applicability to the given short time series. Thus, the presented approaches inspire more sophisticated methods to be used in this thesis, while a more direct application seems unfeasible.

Dresia *et al.* [3] propose a system similar to the one evaluated in this thesis for anomaly detection using a *Feedforward* (FF) forecasting model and a static threshold. It uses the same (proprietary) dataset as this thesis. The model is designed to take physical relations into account. Applying it to other datasets requires substantial knowledge about the underlying processes. While it works well on obvious, direct faults, it performs poorly on more complex fault types.

Siffer *et al.* [13] present a method for AD on time series based on the *Extreme Value Theory* (EVT). While the input to this method is not some kind of (reconstruction) error as for the previous two methods, it can also be the value of a variable itself. The method calculates some sort of threshold. If this is surpassed, an anomaly has been detected. The threshold is computed and updated in a completely unsupervised setting. Similar to the method of Hundman *et al.* [1], it assumes the data to be structured as one long time series.

Another LSTM based approach is described by Wang *et al.* [14]. Here, the *Mahalanobis* distance is used instead of a vanilla reconstruction error in order to compute an anomaly score.

OmniAnomaly is a multivariate time series AD algorithm by Su *et al.* [15]. This approach that can deal with explicit temporal dependence among stochastic variables. The method combines *Gated Recurrent Units*, planar *Normalizing Flows*, stochastic variable connection, and an adjusted *Peaks-Over-Threshold* (POT) method to achieve high performance on real-world datasets.

1.2 Why Unsupervised Reconstruction-based Anomaly Detection?

While Dresia *et al.* [3] present an approach that has been evaluated on the same dataset used in this thesis and shows decent results, it has been built with substantial system knowledge and thus requires a redesign if the system itself is modified or in order to apply the model to another system. A more general approach that makes use of the reconstruction error of an AE could reduce this engineering constraint. It could also improve the performance for the non-trivial relationships not taken into account by the forecasting model. In addition, the prediction part of the AD does not need to be constrained to a static threshold but can be extended using more sophisticated methods, such as the ones presented by the other approaches in Section 1.1.2. If these methods

can be modified to work well on shorter time series, they present a more robust and stable solution for anomaly detection compared to a fixed threshold tuned under optimal conditions.

1.3 Objectives and Requirements

The goal is to design a more sophisticated AD mechanism that can detect less obvious, non-fatal anomalies after the test has finished or even during the test. For security reasons, this system can not replace the fixed thresholds entirely. If the rate of false positives is reasonably low for the test operators (i.e., practically zero), the AD system could support the test abort system to find anomalies earlier and thus increase the safety of the system. In reality, stochastic uncertainties exist, so some false positives have to be tolerated in order to be able to detect anomalies, even when using the conventional red lines method for test aborts. Consequently, the tolerated false positive rate is set to $1\,\%$, which is the ratio between the number of false positives and the number of nominal measurements. Although this false positive rate might sound too high for the test bench operators, it could give insights for the customers on how much they could rely on the results of the test run and whether additional attention is required when gaining insights from the collected data.

1.4 Outline

This thesis is organized into five chapters. In Chapter 2, the theoretical and mathematical background is presented. Chapter 3 explains the methods in greater detail. Chapter 4 presents the results using the test bench dataset. Finally, Chapter 5 contains a summary and conclusions. In addition, supplemental material is listed in the appendix.

Chapter 2

Theory

An elementary reconstruction technique is *Principal Component Analysis* (PCA). While this thesis uses AEs to compute a reconstruction error, PCA helps to understand the dataset and introduces the concepts of *encoding* and *decoding* data. In Section 2.1, the mathematical background for PCA is explained. This is followed by the central concepts and ideas of AEs in Section 2.2.

2.1 Principal Component Analysis

Principal component analysis (PCA) is a technique that can be used to reduce the dimensionality of a dataset. The following section summarizes PCA as described by Jolliffe and Cadima [16].

2.1.1 Calculation of Principal Components

Principal components are linear combinations of the variables that capture the maximum amount of *variance* in a dataset. They are vectors forming an *orthogonal coordinate system*. In order to find principal components alongside their explained variance, PCA uses basic linear algebra. Usually, they are calculated with the help of the *eigenvector-eigenvalue problem* as presented in Equation 2.1, using the *sample covariance matrix* S associated with the dataset. In order to have a well-defined solution to this problem, the vectors are restricted to *unit-length* [16], i.e., $a^T a = 1$.

$$\{(a, \lambda) | Sa = \lambda a\} \tag{2.1}$$

These *eigenvectors* a_k are sorted in descending order by the magnitudes of their corresponding *eigenvalues* λ_k. The principal components can be obtained by multiplying the

corresponding *eigenvectors* on the right of the data X. However, some authors call the *eigenvectors* themselves principal components.

2.1.2 Dimensionality Reduction

If the dataset X should be reduced to a dimension k, then using the first k principal components results in the best representation of *variance* in the dataset. When *normalized eigenvalues* are used such that their sum is 1.0, they correspond to the *explained variance* of the corresponding principal component. With a dataset X, *transformation matrix* T containing the first k sorted *eigenvectors* as columns, and the reduced dataset Y, the following formula is used to reduce dimensionality with PCA:

$$Y = TX \tag{2.2}$$

The reversed operation is:

$$X = T^{-1}Y \tag{2.3}$$

In real systems, the input dataset should be standardized or normalized to avoid the dominance of high values over smaller ones. This standardization or normalization has to be reversed for the inverse transformation, too. PCA does not require the variables to be normally distributed. While PCA is relatively inexpensive in terms of computational complexity, its limitation to linear operations causes trouble when reducing data that has nonlinear behavior.

2.2 Auto-Encoders

Auto-Encoders constitute a nonlinear extension of PCA. Instead of using deterministic linear operations, they fall under the scope of *Neural Network*s (NNs). This section briefly introduces the AE concept along with some NN basics. An AE is an NN that is trained to attempt to reproduce its input [17]. AEs learn representations unsupervised. They have many applications such as dimensionality reduction, feature extraction, denoising, or anomaly detection [18].

2.2.1 Terminology

An *Auto-Encoder* is a special form of NN. This section introduces the concepts and terminology relevant for AEs although some of them are actually valid for NNs in general.

Structure

Typically, AEs consist of an *encoder* and a *decoder*. The n-dimensional input is reduced to a z-dimensional *latent* vector and then extracted back to an n-dimensional output vector. If $z < n$, the latent layer is called *bottleneck*. A compressing AE is depicted in Figure 2.1.

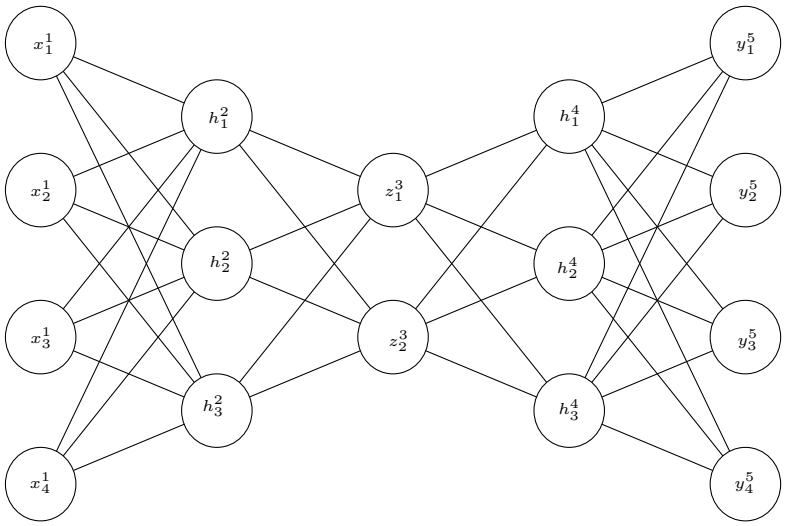


Figure 2.1: A *Feedforward Auto-Encoder* with five layers, confer to Robertazzi and Shi [19]

Loss Function

Like all NNs, AEs need to be trained before they can be used to predict. In order to define what an optimally trained model is, a *loss function* is required. The model training

optimizes the model parameters such that the loss function is minimized. In this thesis, the reconstruction error in form of the *Mean Squared Error* (MSE) is used:

$$\mathcal{L}(x, \hat{x}) = \sqrt{\sum_i (x_i - \hat{x}_i)^2} \tag{2.4}$$

The subscript i denotes the ith component of the feature vector x and its reconstruction $\hat{x}$.

Regularization

The term *regularization* refers to the techniques that are required to force a model not to *overfit* on the training data. Overfitting occurs when a model does not learn the underlying features but fits well to the noise of the training data. If an AE overfits, this can result in learning the *identity function*, thus copying the input as is to the output without any sort of processing. The reconstruction error then will be zero for all input values.

Feedforward

In this thesis, *Feedforward Auto-Encoder*s (FF AEs) are used to compute a reconstruction error for AD. A *Feedforward* model uses only fully connected layers. An example of this form of AE is shown in figure 2.1. In an FF AE, each neuron consists of a linear layer with the following operation

$$y = Ax + b \tag{2.5}$$

where A is the weight matrix and b is the bias vector. In addition, the output is passed through an activation function in order to introduce nonlinearities.

Activation Functions

An activation function is a non-linear function $\mathbb{R} \to \mathbb{R}$. Here, several common activation functions are presented. These are used to modify the output of nodes:

$$\text{linear}(x) = x^1 \tag{2.6}$$
$$\text{relu}(x) = \max\{0, x\} \tag{2.7}$$
$$\text{leakyrelu}(x) = \max\{\alpha x, x\} \qquad 0 < \alpha < 1 \tag{2.8}$$
$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.9}$$
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.10}$$

Figure 2.2 shows the plots of the used activation functions. Some activation functions like *tanh* or *sigmoid* show a relatively high slope around the origin while being flat for extreme values. This results in low gradients for values far from the origin, which can impose difficulties for optimization algorithms. *Leaky-ReLU* as well as *ReLU* are not differentiable in the origin, which is a problem that has to be handled in software (such that the gradient is set to a value for a zero input).

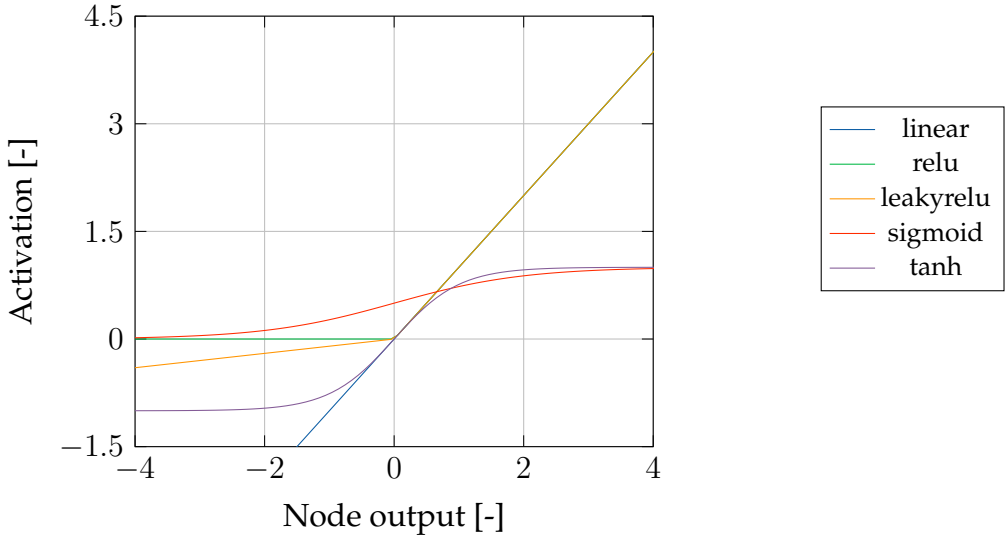


Figure 2.2: Common activation functions

[1] A *linear* or *identity* function does not qualify as activation function. However, it can be used to measure the impact of an activation function on the model performance.

Chapter 3

Methodology

This chapter elaborates on the details of the *P5* LOX dataset in Section 3.1. The preprocessing of this dataset is presented in Section 3.2. Section 3.3 deals with the application details of AEs. The training process is elaborated in Section 3.3.4, followed by the method of *Hyper-Parameter* (HP) tuning in 3.4. The AD methods that have been evaluated in this thesis are described in Section 3.5.

3.1 P5 LOX Dataset

The following section describes the relevant details of the data generation process. For this thesis, synthetic data from the rocket engine test facility P5 in Lampoldshausen has been used. This data has been generated in advance by Dresia *et al.* [3] using the simulation software *EcosimPro*. As this process is not part of the thesis work, refer to Dresia *et al.* [3] for more details about the generation. Some Information about the system can be found in Section 3.1.1 while Section 3.1.2 gives an overview of the data generation process. Details and variables of the data are explained in Section 3.1.3. As synthetic anomalies have been simulated as well, these are presented in Section 3.1.5. As it is common practice in the space sector, the data cannot be published as a whole. Anyway, detailed analysis can be presented using a generic sequence authorized for publication by the test bench operators.

3.1.1 P5 LOX System

The *Liquid Oxygen* (LOX) system of the P5 test facility at DLR Lampoldshausen has been modeled in advance by Dresia *et al.* [3] and provides a fair amount of experimental data from previous test campaigns. Under these conditions, the LOX system represents a candidate to evaluate the potential of ML based AD in rocket engine test facilities [3].

The system is situated above the engine in order to provide the engine with oxidizer pressurized according to requirements defined by the operators. A schematic version

Figure 3.1: Testbench P5

of the system is provided in Figure 3.2. The LOX system contains two separate circuits. The upper part is a pressurization circuit. With the help of *Gaseous Nitrogen* (N2) and controlled by a *Proportional Integral* (PI) controller, this system regulates the pressure of the LOX tank. The LOX tank is filled with up to $200\,\mathrm{m}^3$ LOX. It is the starting point of the LOX circuit. The tank is followed by a series of sensors and pipes that terminate in the engine interface. Refer to Table 3.1 for details on the sensors and actuators.

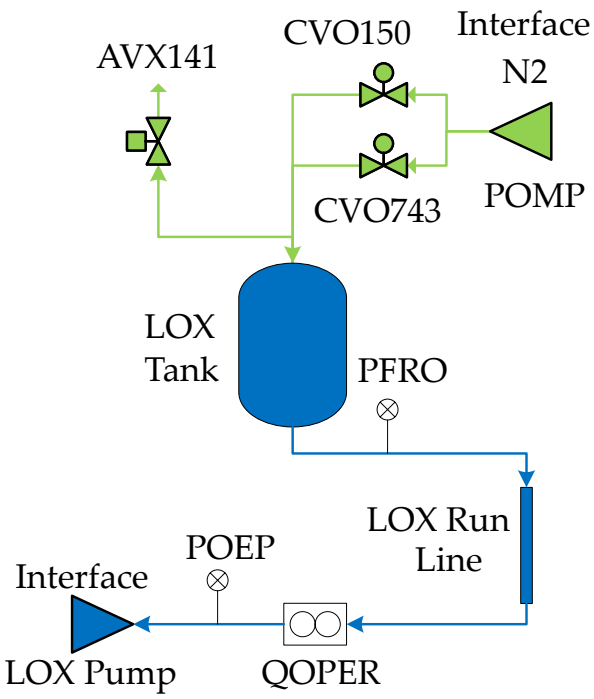


Figure 3.2: Simplified scheme of P5 LOX system, according to Dresia *et al.* [3]

Table 3.1: Subsystems of the LOX system of the P5 test bench

N2 circuit	
`POMP`	Pressure of the pressure regulator between N2 tanks and the pipes
`AVX141.pos`	Ventilation valve
`COOX141.Amp`	Command for `AVX141`
`CVO150.pos`	Large N2 pressure regulator valve
`CVO743.pos`	Small N2 pressure regulator valve
`CVO150Arc`	Reference tank pressure
`POMP`	N2 supply pressure
LOX circuit	
`PFRO`	tank pressure in bar
`Tank.Vl`	Volume of liquid in tank
`QOP2MA`	LOX mass flow rate
`POEP`	engine interface pressure

3.1.2 Simulator

The P5 LOX system has been simulated using the software *EcosimPro* along with the *European Space Propulsion System Simulation* (ESPSS) library. The software provides valves, sensors, pipes, tanks, and other components to simulate complex fluid systems [20].

3.1.3 Simulation Output

The simulation output contains sensor and actuator data. The simulation produces additional outputs that lack a physical representation and are excluded from any analysis in this thesis. The output is grouped into independent time series, also called *runs*. Each *run* is started with a randomized `LOX` tank volume. The control sequences for the actuators are also randomized using realistic assumptions [3]. The data is *time discrete* with an interval of 1 second between measurements.

3.1.4 Nominal

15000 runs have been generated without modifications to the generation process. This means, the simulation uses a structure similar to Figure 3.2. This data represents the nominal part and is used for training and validation.

3.1.5 Faulty

For a separate part of the data, different parts of the system have been modified. These simulated faults can be grouped into sensor faults and system faults. The faults taken into account are sensor offsets, drifting sensors, frozen sensors, and leaks. These modifications occur at a random point in the run, so that the start is essentially a nominal run. Once a fault has been introduced into the system, it will stay active until the end of the run. For each anomalous run, there is exactly one sort of anomaly present. The faults can be grouped into two distinct groups.

Sensor Faults Here, the values of the `POEP` sensor have been modified to model a sensor failure. Figure 3.3 illustrates these three types of faults.

- **Sensor Frozen**
  The value of one of the sensors in the system stops being updated as soon as the fault is introduced.
- **Sensor Offset**
  Upon introduction of this fault, a random constant is added to the value of the sensor for each reading. The random offsets range from 0.5 to $1.5\,\mathrm{bar}$.
- **Sensor Drift**
  An increasing value is being added on top of the reading of a sensor. This drift rate can range from 2 to $4\,\mathrm{mbar}\,/\,\mathrm{s}$.

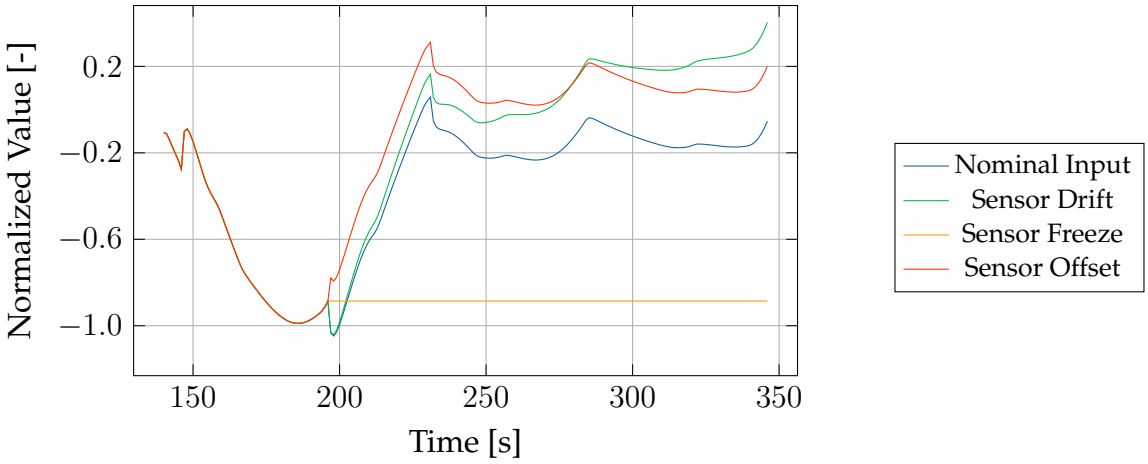


Figure 3.3: Fault influences on the interface pressure POEP for the generic sequence

System Faults In addition to the faulty sensors, faults can occur in pipes and valves. A small selection of the possible system faults has been generated for the analysis. The Figures 3.4, 3.5, and 3.6 give an impression of the influence of these faults on the system.

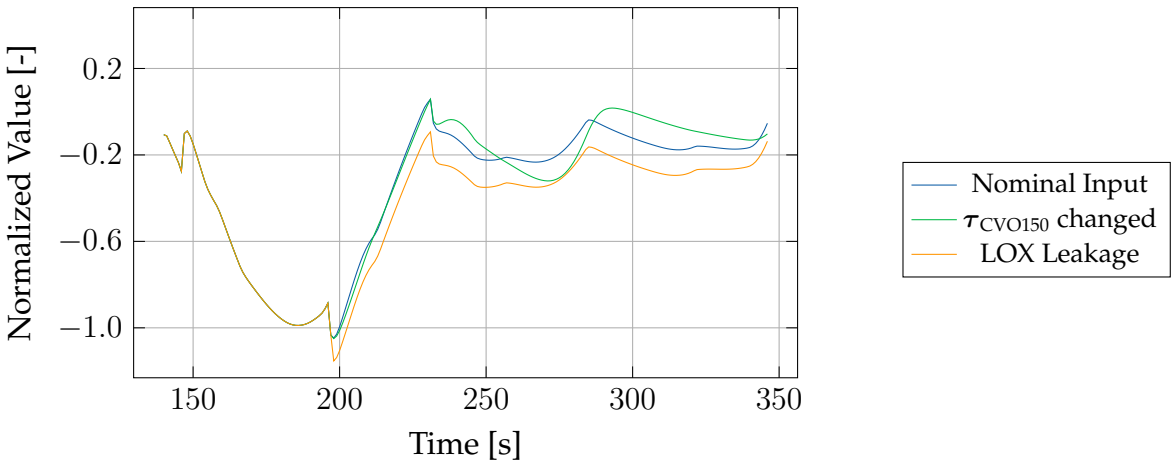


Figure 3.4: POEP for the generic sequence

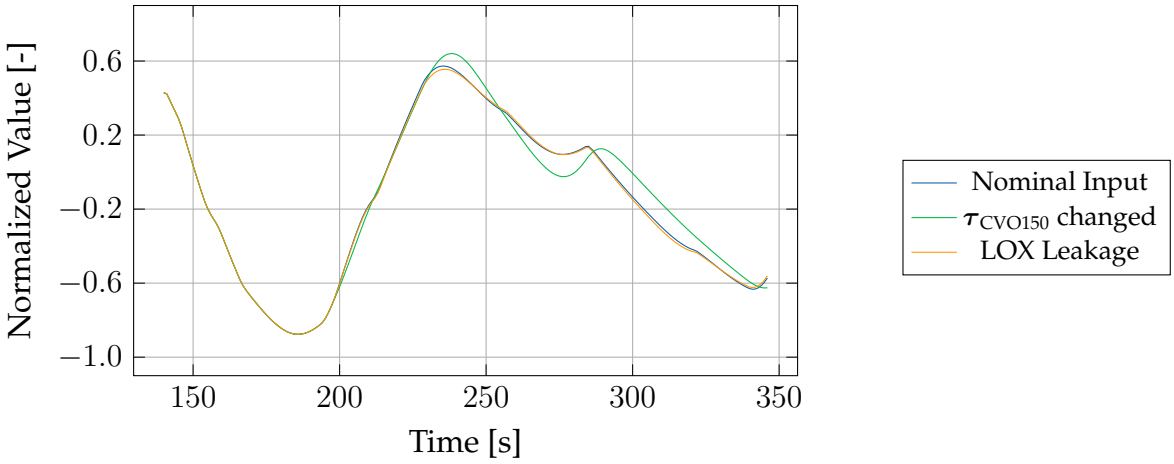


Figure 3.5: PFRO for the generic sequence

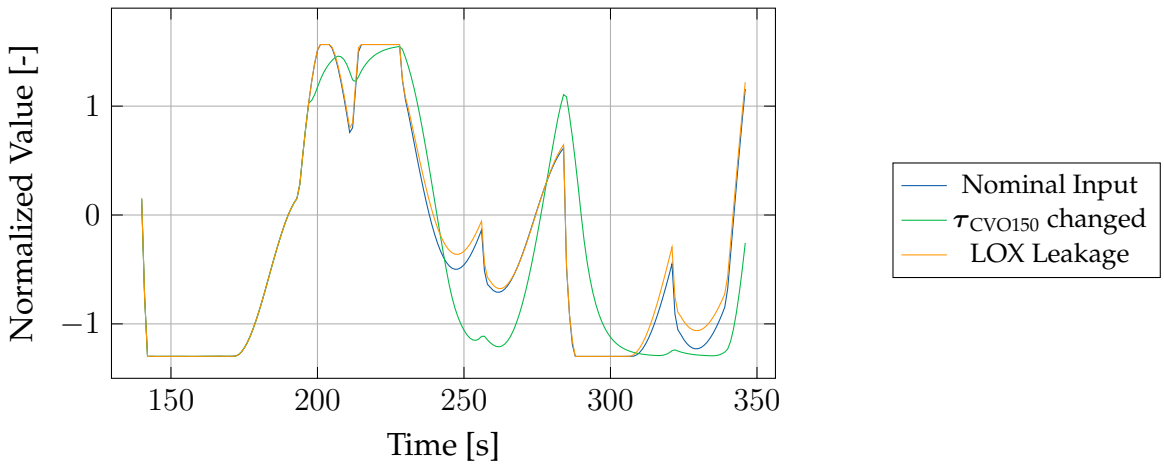


Figure 3.6: CVO150.pos for the generic sequence

- τ_{CVO150}
  The fault consists of changing the average time constant of the first order transfer function in the simulation of moving the CVO150 valve. CVO150 is controlled by a PI controller which takes the difference between a desired LOX tank pressure and PFRO. Thus, the control process will be modified, and PFRO and, subsequently, POEP change.
- **LOX Leakage**
  An extra outlet with an average mass flow of $11.2\,\mathrm{kg\,s^{-1}}$ is simulated in the LOX line in order to simulate a leak in this system. This reduces the pressure POEP at the end of the LOX line. To some extent, the tank pressure PFRO is also getting reduced. In consequence, with the help of the controller, this also changes CVO150.

3.2 Preprocessing

Before the data can be used by a model, it has to be read and scaled. Details on this will be described in Section 3.2.1. In addition, Section 3.2.2 describes how and why *sliding windows* are created out of the dataset.

3.2.1 Scaling

Three different types of scalers will be used in order to move the different features into a similar range. This is necessary in order to avoid situations where one feature is orders of magnitude greater than the other features and thus dominates the computation of the MSE. The scalers are described as implemented in *Scikit Learn* 1.2.0 [21].

- **Standard Scaler**
  Assuming a standard distribution for the values of a feature, a *standard scaler* removes the mean and scales to unit variance. The *standard score* of a sample x is calculated as

$$z = \frac{x - u}{s} \tag{3.1}$$

  using the *sample mean* u and *sample standard deviation* s.
- **MinMax Scaler**
  This scaler does not make assumptions about the distribution of the data. It moves and scales the data such that any value lies between a minimum and a maximum. The transformation is provided by

$$X_{\text{std}} = \frac{X - \min\{X\}}{\max\{X\} - \min\{X\}} \tag{3.2}$$

$$X_{\text{scaled}} = X_{\text{std}} \cdot (\partial_+ - \partial_-) + \partial_- \tag{3.3}$$

  using a sample X, a desired minimum ∂_- and a desired maximum ∂_+. When using a *MinMax scaler* in models, the output layer should contain a suitable *activation function*. In the case of $\partial_- = 0$ and $\partial_+ = 1$, this will be the *sigmoid* function. When using $\partial_- = -1$ and $\partial_+ = 1$, a *tanh activation function* should be used. This will limit the output values to the range of the scaled input values.
- **Robust Scaler**
  The *MinMax scaler* and, to some extent, the *standard scaler*, scale a dataset poorly if it contains *outliers*. These are values that have a large distance from the high-density part of the data distribution. They can change the mean significantly and force a *MinMax scaler* to use a large range. To be less affected by these outliers, the *robust scaler* can be used. Instead of removing the mean, it removes the median. In addition, it uses the *interquartile range* to scale the data into a range. The *interquartile range* is the range between the first and third *quartiles* of a distribution. The

robust scaler computes as

$$z = \frac{x - m}{b - a} \tag{3.4}$$

with median m, first *quartile* a and third *quartile* b.

3.2.2 Sliding Windows

The time series data used in this thesis contains temporal dependencies. The most obvious one is the relation between `PFRO` and `CVO150.pos`. A PI controller commands `CVO150.pos` using the error between desired tank pressure and `PFRO`. By using the plain, scaled variables, a FF model would not be able to learn this kind of dependency. The usual workaround is to use *sliding windows*. For any time $t \geq k$, t and the values from $t - 1$, $t - 2$, ..., $t - k$ are being stacked. Assuming equal times of $1\,\mathrm{s}$ between time steps, this gives the model the opportunity to take past values into account. The span $k + 1$ is called *sliding window size* in the following. A *sliding window size* of one is equivalent to skipping the stacking step.

3.3 Details on Auto-Encoders

While the foundations of AEs are described in Section 2.2, this section presents methods that are applied on AEs in this thesis. These are greater details on regularization in Section 3.3.1 and the role of a bottleneck in Section 3.3.2 as well as details on the activation function in Section 3.3.3.

3.3.1 Regularization

Kukaka *et al.* [22] provide an overview on regularization methods that are summarized in this section. A large variety of different methods exist to regularize a model. Regularization enables a model to perform well on unseen data. Conventionally, regularization is understood as a modification of the *loss function*, i.e., the addition of a regularization term to it. A common term is the $L2$ term. This sort of regularization can be called *direct regularization* in contrast to *indirect* methods. *Indirect regularization* includes the choice of the *loss function*, *dropout* on the nodes, *batch normalization*, a *bottleneck*, as well as the choice of *optimization* and the *activation function*.

3.3.2 Bottleneck

Yong and Brintrup [23] elaborate whether a *bottleneck* is required for AD using AEs. The reasoning is summarized in this section. Specifically for AEs, regularization is important to prevent *overfitting* or, in extreme cases, learning the identity function. The model would copy the inputs directly to the outputs without extracting meaningful features.

In the use-case of *Anomaly Detection*, this would mean that the reconstruction error becomes zero for any input value, and thus, no anomalies could be detected.

Usually, a *bottleneck* is used to avoid this undesirable behavior. This means that the latent dimension is smaller than the input/output dimension. This should prohibit the AE in learning the identity function instead of relevant features. However, the paper proposes that in the case of AD, non-*bottleneck* AEs result in better detection of anomalies and not in learning of the identity function. Options to remove the *bottleneck* include *skip connections* and *over-parametrization* of the latent layer. There is no proof that a *bottleneck* is necessary to avoid learning the *identity* function. While counter-intuitive, the other regularization measures, such as *random initialization*, non-linear *activation functions*, and the *ADAptive Moment estimation* (ADAM) optimizer, make sure the model learns the relevant features.

3.3.3 Output Activation

The result of each node is often passed through an *activation function* such as *sigmoid*, *tanh*, *relu* or *leakyrelu* [24]. These add nonlinearities to the models. For the *activation functions* in the output layer, different considerations become relevant. As AEs have to recreate the input, their output range should correspond to the input range. Thus, depending on whether the input has been scaled to a range, activation functions can be the *linear* function in the case of real-number input, the *tanh* function for input values between -1 and 1 and the *sigmoid* function for values between 0 and 1.

3.3.4 Training

The models have been implemented using *Pytorch Lightning* (PL). The frameworks used are described in greater detail in the appendix (A.1). In order to select well-working combinations from the vast amount of possible HP combinations, a HP study is being run using *Optuna* (details in Section A.1) for each model architecture. The study is set up to search for the highest *True Positive Rate* (TPR) at a *False Positive Rate* (FPR) of $1\,\%$. The FPR has been set to a fixed value as *False Positive*s (FPs) are highly undesired in the test bench operation. To reduce the training time, the *loss* is monitored. If the *loss* does not improve significantly within a validation period, the *learning rate* is reduced by a factor. In addition, *early stopping* is activated: If the *loss* does not improve for several validation periods, the training is stopped.

3.4 Hyper-Parameter Tuning

While the training could be run directly without further thought about the model configuration, this will seldom produce the desired results. This chapter deals with the details of the model configuration, especially an automatic method to find well-working configurations.

3.4.1 Hyper-Parameters

In order to understand what parts of a model and its training have to be configured, some terms have to be defined. There are two types of parameters in machine learning: *Model parameters* are set during the training process. *Hyper-Parameter*s (HPs) have to be set in advance [25]. For the latter, different values can have a huge effect on the performance of the model. The possible values of all HPs combined form the search space.

3.4.2 Random Search

In order to find well-performing configurations, the search space is being explored by taking random samples [25]. In order to avoid unnecessary computations and thus shorten the search time, some assumptions on the ranges for HPs are necessary. This thesis uses the *random search* method. While this approach takes longer than methods that take earlier results into account [25], it is very easy to set up as the only information required is the search space and an optional random seed. Because no prior state is required, *random search* could easily be parallelized [25]. In addition, complex search algorithms that require many parameters can lead to trouble when these have not been set carefully. As a result, they might underperform or yield misleading results.

3.4.3 Optimization of the Hyper-Parameter Tuning Runtime

In order to avoid spending a lot of time on the training of under-performing models, the search space should be narrowed after training a representative number of models [25]. If necessary, new search spaces can be explored too. The initial search space should be defined using some prior experiments and constraints.

3.4.4 Hyper-Parameters and Their Ranges for the Feedforward Auto-Encoder

A first step in the process of HP tuning is to train some models with different HPs. In this thesis, these sample models have been chosen taking into account the technical limitations in terms of runtime and memory as well as some heuristics based on the knowledge gained from Dresia *et al.* [3]. The actual search for well-working HPs is performed in two rounds. In a first round, relatively broad ranges for many parameters are used, as presented in Table 3.2. While the choice of an *optimizer* is also a *Hyper-Parameter*, it has been set to the ADAM optimizer. After training 181 models selected by random search, the study is interrupted by manual intervention.

Evaluating the ten best models of the first round is then used for a redefinition of the search space for another round of 144 models. Some of the HPs can be set to a constant value as they do not seem to influence the model performance enough to justify spending search time on them. Thus, the updated ranges are presented in Table 3.3.

Table 3.2: HP ranges for the first round

HP	Type of choice	range
Related to training		
Learning Rate	logarithmic	$10^{-3} \dots 10^{-1}$
Number of Epochs	integers	$5 \dots 50$
Batch Size	selection	512, 1024, 2048
Learning Rate Decay Factor	logarithmic	$0.05 \dots 0.2$
Learning Rate Patience	integers	$0 \dots 2$
Learning Rate Threshold	logarithmic	$0.01 \dots 0.2$
Early Stopping Minimal Delta	logarithmic	$0.01 \dots 0.1$
Early Stopping Patience	integers	$5 \dots 15$
Related to the model		
Latent Dimension	integers	$2 \dots 128$
Activation Function	selection	*ReLU, Tanh, LeakyReLU, Sigmoid*
Number of Layers (hidden layers)	integers	$0 \dots 3$
Neurons per (hidden) Layer	integers, logarithmic	$2 \dots 256$
Dropout	float, relevant when more than 0 hidden layers	$0.0 \dots 0.2$
Features	selection from all features	$3 \dots 10$ features
Sliding Window Size	integers	$1 \dots 30$
Scaler Type	selection	*Standard Scaler, Robust Scaler, MinMax Scaler(-1, 1), MinMax Scaler(0, 1)*

Table 3.3: HP ranges for the second round

HP	Type of choice	range
Learning Rate	logarithmic	0.001 ... 0.08
Number of Layers (hidden layers)	integers	0 ... 1
Neurons per (hidden) Layer	integers, logarithmic	150 ... 250
Features	selection from all features	4 ... 10 features
Number of Epochs	integers	11 ... 49
Sliding Window Size	integers	4 ... 27
Scaler Type	selection	*Standard Scaler, Robust Scaler, MinMax Scaler(-1, 1), MinMax Scaler(0, 1)*
Batch Size	selection	512, 1024, 2048
Latent Dimension	integers	32 ... 128
Activation Function	selection	*Tanh, LeakyReLU, Sigmoid*
Dropout	float, only relevant for AEs with more than 0 hidden layers	0.0 ... 0.2
Learning Rate Decay Factor	fixed	0.1
Learning Rate Patience	fixed	2
Learning Rate Threshold	fixed	0.05
Early Stopping Minimal Delta	logarithmic	0.01 ... 0.09
Early Stopping Patience	fixed	13

3.5 Thresholding Methods

As an *Auto-Encoder* (AE) does not predict anomalies, some algorithm is required to proceed from the results of the AE to predicted labels. According to the *Anomaly Detection* (AD) theory, the reconstruction error of the AE can be used as anomaly score directly [4]. For this thesis, the *Mean Squared Error* (MSE) (L2 norm) of the difference between the *sliding windows* of original data and reconstruction is being used as input for AD methods. For the evaluation of AD methods, the structure of the data is relevant. The training data is purely nominal. A total of 1250 runs contain constructed, labeled faults. Thus, it is possible to tune AD methods in a supervised manner.

3.5.1 Static Thresholding

Static thresholding is the most basic method for performing AD from the reconstruction error of the AE. Using MSE as input, any value that exceeds a certain threshold is flagged as anomalous. Thus, the corresponding measurement has to be anomalous. Depending on the requirements of the task, this threshold can be set to a specific value.

- **$0\,\%$ False-Positive Rate on Validation Data**
  The threshold can be set entirely unsupervised, taking into account that the training (and thus, the validation) data are purely nominal. The idea of this method is to set the threshold high enough so that no measurement in any validation run is flagged as anomalous. Depending on how well the model learns to reconstruct the input, the reconstruction error on nominal data can still be quite high from time to time.
- **$1\,\%$ False-Positive Rate Faulty Data**
  Taking into account that the faulty data is labeled, the threshold can be set with the help of the *Receiver Operating Characteristic* (ROC) curve of the reconstruction errors on the faulty data against the labels [26]. Here, the requirement on the FPR should be taken into account in order to set the threshold. The corresponding TPR can be used as a score to compare the performance of different AEs.

3.5.2 Other Anomaly Detection Methods

The static thresholding methods suffer from two issues: They require a very controlled environment with the same mean and variance of the nominal reconstruction error as used for the threshold selection. In addition, the distribution of the reconstruction error for nominal values must not change during a run.

Parametric Thresholding

Hundman *et al.* [1] propose an AD method for dynamic environments. *Parametric Thresholding* is based on the idea that a long time series of data can show slow changes in mean and variance in the reconstruction error, which are not anomalies. In a phase of

relatively low mean and variance, a sudden but still low-level increase in the reconstruction error hints at the occurrence of an anomaly. In other parts of the time series, even the mean and variance of nominal data can exceed this value.

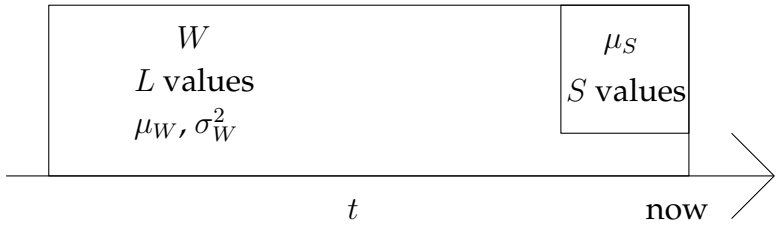


Figure 3.7: Overview of the variables used in parametric thresholding

According to Hundman *et al.* [1], two sliding windows are being calculated for each value. The long one, called W, is used to find the mean μ_W and variance σ_W^2 of the current phase of the time series. The short window is used to smooth the reconstruction errors at the moment and find a mean μ_s. Then, the anomaly likelihood L is calculated as

$$L = 1 - Q\left(\frac{\mu_s - \mu_w}{\sigma_W^2}\right) \tag{3.5}$$

using the tail probability, also called Q-Function. The Q-Function describes the probability for a random variable that its value will exceed a threshold [27]. An anomaly is detected if

$$L \geq 1 - \epsilon_{\text{norm}} \tag{3.6}$$

using an anomaly threshold ϵ_{norm}. In the paper of Hundman *et al.* [1], a long window of length $l_W = 2100$, a short window of length $l_s = 10$ and $\epsilon_{\text{norm}} \in \{0.01, 0.0001\}$ have been used.

However, the data used for this thesis has a form that does not allow for these values to be used. The typical run length is of the order of several hundred seconds. Every run is independent of the others. That said, some modifications have been made to make an implementation possible. An overall mean and variance can be calculated using the nominal data or the nominal parts of the faulty runs. For the start of the runs, the mean and variance to be used in the Q-Function will be calculated using a weighted average of a sliding window from start to the current value with the overall mean and variance. In addition, the length of the long window is decreased to a size that is smaller than most runs.

Peaks-Over-Threshold

As the original version for *parametric thresholding* requires a sliding window size that is longer than the time series in the *P5* LOX dataset, the method might show weak

performance. As an alternative, another statistical method is presented in this section. Siffer *et al.* [13] propose POT, a method designed to be more stable in more open environments. The approach is based on EVT. EVT is a method to find a distribution for extreme events. This tail of a distribution can also be fit using the POT approach [13]. A sub-variant of this method adjusts the anomaly threshold dynamically.

Stationary This method calculates anomalies by taking the whole input at once. From the values, an initial threshold t is computed. Any value greater than the threshold t is selected. The threshold is then subtracted from the value. This selection is called the *peaks set*. The *peaks set* is then fitted on a *Generalized Pareto Distribution* with the help of the *Grimshaw* trick, yielding $\hat{\gamma}$ and $\hat{\sigma}$. *Grimshaw's trick* is a method to reduce a two-variable optimization problem (here the fit of two parameters for the distribution) to a single-variable equation. The mathematical details can be found in the original paper by Siffer *et al.* [13]. The quantile or anomaly threshold can then be calculated with

$$z_q \simeq t + \frac{\hat{\sigma}}{\hat{\gamma}} \left(\left(\frac{qn}{N_t} \right)^{-\hat{\gamma}} - 1 \right), \tag{3.7}$$

using the threshold t as defined above, the "desired probability" q, n as the total number of observations, and N_t as the number of *peaks*, i.e., the size of the *peaks set* from above. Any value above the threshold z_q is then flagged as anomalous.

Streaming For an initial set, thresholds t and z_q are computed. Then, for any further value, if it exceeds z_q, it is added to the anomalies. Otherwise, if it exceeds the threshold t, it (minus the threshold t) is added to the *peaks set* and used to update the *Generalized Pareto Distribution* and, subsequently, the threshold z_q.

Streaming with Drift For "normal" streaming, no drift in the data is taken into account. If the data itself is dynamic, this will result in unsatisfactory detection. To counter this issue, the algorithm is executed on some sort of sliding window. A mean M is calculated from nominal initialization data. In an initialization loop, this mean is updated using the next n values. In each step, the mean is subtracted from the current value, and the result is added to the POT initialization set. After the initialization loop, the thresholds t and z_q are calculated with the stationary POT algorithm. For any following value, the current mean is subtracted. If the result is greater than z_q, it is added to the anomalies, and the current mean becomes the next mean. If the result is greater than t, it (minus the threshold t) is added to the *peaks set*. From the updated *peaks set*, the *Generalized Pareto Distribution* and, subsequently, the threshold z_q are updated. In this case as well as when the value is less than t, the mean is updated from a window of (unmodified) values shifted by one.

Chapter 4

Results and Discussion

This chapter presents an evaluation of the AD methods presented in the previous chapters. Section 4.1 defines the scores and metrics used later to evaluate and compare the AD methods. Prior to the actual analysis of the AD methods, the data itself is analyzed with the help of PCA in Section 4.2. The model resulting from the HP study is presented in Section 4.3. Section 4.3.1 compares this model for different forms of regularization. This is followed by the evaluation of the *bottleneck* in Section 4.3.2. Section 4.4 compares the effects of different *thresholding* methods. The key results are summarized and discussed in Section 4.5.

4.1 Evaluation Metrics

When an AD method is evaluated on labeled data (in this case, on the test data containing faults), the performance of the algorithm can be measured with several scores, depending on the requirements. The scores used in this thesis will be described in this section as defined by Fawcett [28]. For the presented scores, the terms *True Positive* (TP), FP as well as *True Negative* (TN) and *False Negative* (FN) are defined as the number of expectation-prediction pairs that fulfill the following definitions:

- **TP**
  The expected label and the predicted label are both positive. This is also known as *hit*.
- **FP**
  The expected label is negative, while the predicted label is positive. This is also known as *false alarm*.
- **TN**
  The expected label and the predicted label are both negative, which is also known as *correct rejection*.

- **FN**
  While the expected label is positive, the predicted label is negative. This is also known as *miss*.

Confusion Matrix These terms can be grouped into a matrix as presented in Table 4.1 which is then called *confusion matrix*. This matrix contains all the data necessary to compute the following scores.

Table 4.1: Confusion matrix

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

The confusion matrices for the methods and fault types can be found in the appendix in Section A.3.

Scores The confusion matrix lacks the simplicity of a score that can be compared easily. In order to evaluate the performance of the AD model, the following scores are used:

- **False Positive Rate**
  The *False Positive Rate* (FPR) is the probability of a *false alarm*. It is estimated as
  $$FPR = \frac{FP}{FP + TN}. \tag{4.1}$$
- **True Positive Rate/Recall**
  The *True Positive Rate* (TPR) is the probability of *hit* or *hit rate*. It is estimated as
  $$TPR = \frac{TP}{TP + FN}. \tag{4.2}$$
- **Precision**
  The *precision* is estimated as
  $$precision = \frac{TP}{TP + FP}. \tag{4.3}$$
  In other words, it measures how many of the *predicted positives* are *actual positives*.
- **F1 Score**
  The *F1 score* is calculated as the harmonic mean of precision and recall:
  $$F_1 = 2\frac{precision \cdot recall}{precision + recall} = \frac{2\,TP}{2\,TP + FP + FN} \tag{4.4}$$
  The only number that is not taken into account by the *F1 score* is the number of *True Negatives*. Other than that, this score combines all information from the *confusion matrix* so it creates an overall impression of a classifier's performance.

Receiver Operating Characteristics Curve The ROCs are traditionally defined for methods that contain a static threshold. Then, TPR and FPR are evaluated for different thresholds to calculate their relationship. For a quantative evaluation of the performance, the TPR is plotted over FPR. This results in the ROC curve as illustrated in Figure 4.1.

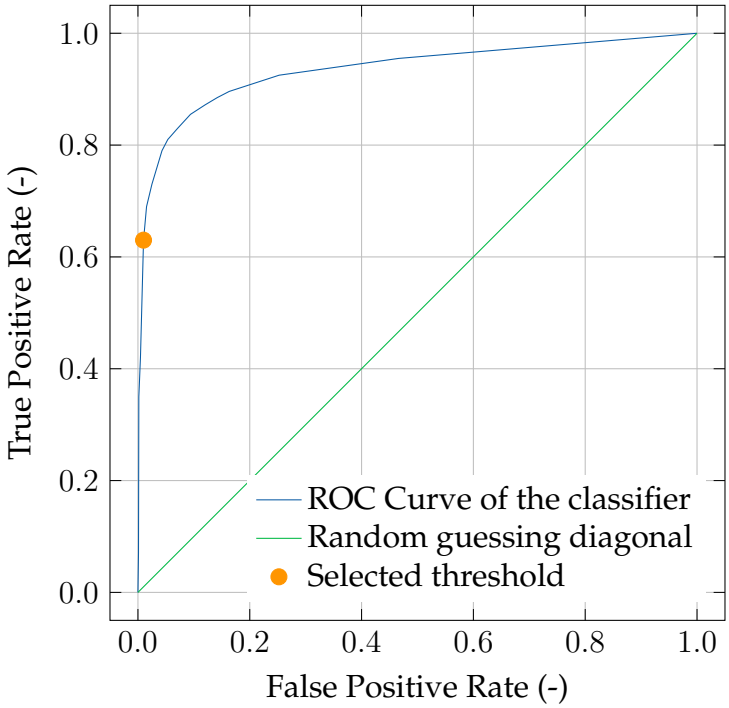


Figure 4.1: The ROC curve for a fictional classifier

4.2 Principal Component Analysis

As the given number of variables is already low, another reduction using PCA does not seem necessary. However, the *correlation matrix* as depicted in Figure 4.2 can help to understand the dependencies of the variables on each other.

Within the N2 circuit, some striking negative correlations exist. First, the pressure `POMP` is strongly negatively correlated with `CVO150.pos` and `CVO743.pos`, to some lesser extent also with `CVO150Arc`. As open inlet valves in the N2 circuit increase the velocity at the valves, its static pressure is reduced. `CVO150.pos` and `CVO743.pos` are often controlled in a similar way as they both control the N2 flow. As `CVO150Arc` is the desired value for the PI controller of `CVO150.pos`, it correlates to the valve position. However, `CVO150Arc` correlates even more with the pressures `PFRO` and `POEP` in the LOX circuit as the valves should regulate the `PFRO` pressure.

`AVX141.pos` and `COOX141.Amp` are grouped together as they correspond to a valve

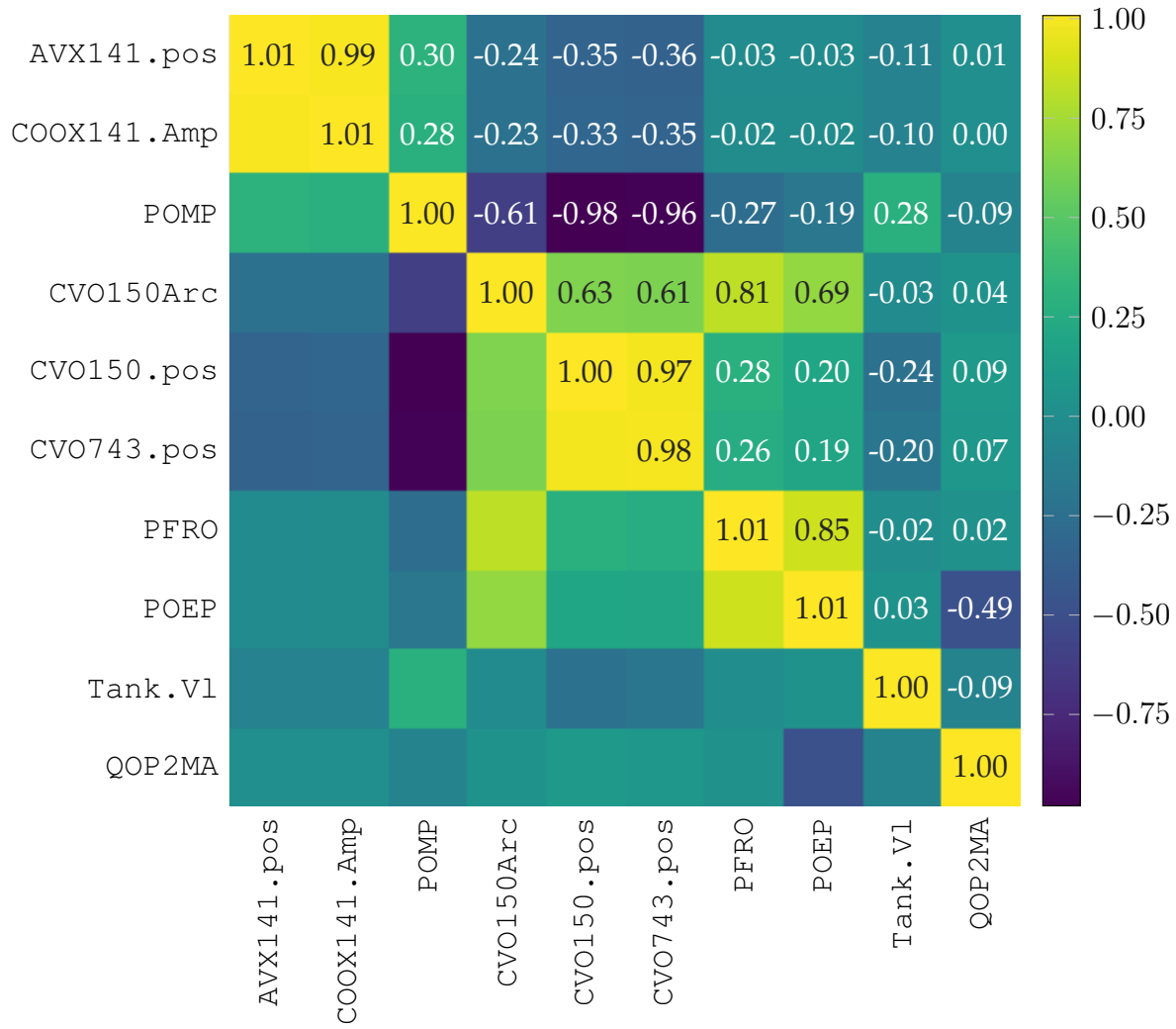


Figure 4.2: Correlation matrix of the features. Values greater than 1 exist due to numerical inaccuracies

and its command. `AVX141.pos` and `COOX141.Amp` show a positve correlation with `POMP` and a negative correlation with `CVO150.pos`, `CVO150Arc`, and `CVO743.pos`. A condition of the test series generation is to not open the ventilation valves when the `CVO150` and `CVO743` valves are open. Thus, the static pressure in `POMP` must be high during ventilation.

The volume of LOX in the tank, `Tank.Vl`, is reduced by increasing the amount of N2 in the upper part of the tank. Opening the `CVO150` and `CVO743` valves will increase the pressure in the N2 circuit and thus press LOX out of the tank. On the contrary, releasing N2 using the `AVX141` valve allows the LOX to stay in the tank which leads to a slightly negative correlation between these features. `QOP2MA` describes a mass flow rate and `POEP` the static pressure nearby. As the static pressure decreases with a higher flow of

LOX, a negative correlation exists between both variables.

4.3 Model Architecture

An HP study is performed on the HPs of the FF AEs and its training process. With the help of *random search*, several models that score well have been found. The best model has a TPR of 0.790 for a tolerated FPR of 0.010. Its *Hyper-Parameter*s are listed in Table 4.2.

Table 4.2: HPs of the best model of the HP study

Hyper-Parameter	Value
Latent dimension	110
Sliding Window Size	5
Activation function	tanh
Features	`PFRO`, `POMP`, `CVO150Arc`, `AVX141.pos`, `QOP2MA`, `POEP`, `CVO743.pos`, `COOX141.Amp`, `Tank.Vl`, `CVO150.pos`
Batch size	2048
Scaler type	*Robust scaler*
Initial learning rate	7.6773×10^{-3}
Maximal number of epochs	32

4.3.1 Regularization

This model lacks a bottleneck, but the performance in AD indicates that some sort of regularization takes place in this model. Direct regularization can be evaluated by training the model with and without $L2$ regularization. In another comparison, the activation function of the model is replaced with the identity function in order to demonstrate its effect on the model's performance. For the resulting four combinations, the performance is evaluated using a static threshold set to allow for a FPR of $1\,\%$ on the faulty data. The results are presented in Table 4.3.

Table 4.3: TPs for combinations

	No $L2$	$L2$
With activation function *tanh*	**0.79**	0.77
Without activation function	0.12	0.26

While the $L2$ regularization does not influence the original model much, it increases the performance without an activation function. The models without activation function perform significantly worse than those with a *tanh* activation. This indicates that the choice of the activation function contributes significantly to the regularization.

4.3.2 Bottleneck

Out of the 20 best models from the HP study, the six best models are all expanding in latent space, most of them by a factor of around 2. The best performance of a model with compression is $TPR = 0.720$ with a compression rate of 1.35. The results and some details about their architecture can be found in Table 4.4.

Table 4.4: The 20 best models from the HP study.

Compression	In-Features	Latent Size	Scaler	Activation Function	TPR
0.45	50	110	*Robust*	*tanh*	0.790
0.36	40	110	*Robust*	*sigmoid*	0.789
0.19	24	124	*Standard*	*tanh*	0.788
0.57	25	44	*Standard*	*sigmoid*	0.786
0.61	75	123	*Robust*	*tanh*	0.773
0.65	72	110	*Standard*	*sigmoid*	0.763
1.35	135	100	*MinMax (-1, 1)*	*leaky-relu*	0.720
0.93	96	103	*Robust*	*tanh*	0.719
1.35	153	113	*Robust*	*sigmoid*	0.703
1.39	161	116	*MinMax (0, 1)*	*sigmoid*	0.701
0.64	54	85	*MinMax (-1, 1)*	*tanh*	0.650
1.14	40	35	*MinMax (-1, 1)*	*leaky-relu*	0.647
0.65	80	124	*MinMax (-1, 1)*	*leaky-relu*	0.614
0.79	96	122	*MinMax (-1, 1)*	*sigmoid*	0.592
0.77	99	128	*Standard*	*sigmoid*	0.586
2.48	260	105	*Standard*	*tanh*	0.558
0.75	80	106	*MinMax (0, 1)*	*sigmoid*	0.544
1.54	63	41	*Robust*	*leaky-relu*	0.535
0.72	64	89	*MinMax (-1, 1)*	*leaky-relu*	0.524
1.17	110	94	*Standard*	*sigmoid*	0.519

The *over-expanding* models show on average better performances than their *bottlenecked* counterparts. In addition, they do not compress to a very small *bottleneck* but still to a relatively high latent size. The better performance of *over-expanding* architectures indicates that no *identity function* has been learned by them. Instead, they seem to learn the internal features of the data. The plots of the MSE in Section 4.4 can also hint that the reconstruction error behaves in the expected way: For nominal values, the reconstruction errors are mostly low. They are significantly higher for anomalies. In addition to the *over-expanding* architectures, the absence of the *ReLU* activation function is remarkable. In fact, the *ReLU* activation has been excluded from the second round of the HP study for poor performance.This activation function is very similar to the identity function. Given the results of Section 4.3.1, the *ReLU* function cannot introduce enough nonlinearity into the architecture in order to regularize the model.

4.4 Thresholding

This section presents the results of various thresholding methods applied on the reconstruction error of the AE. The approaches are presented, starting with *static thresholding* and its variations and continuing with statistical methods. In this section, examples for a generic run are shown. The different fault types have been simulated with the same start time and with the same nominal part. For the different thresholding methods, the runs are presented that illustrate their strengths and weaknesses best.

4.4.1 Static Thresholding

For the static thresholding methods, values that surpass a threshold value are flagged as anomalous. For the first two methods, *0 % False-Positive Rate on Validation Data* and *1 % False-Positive Rate on Faulty Data*, the MSE is used without modification. The later methods change the MSE for some parts of the runs. The methods are used as described in Section 3.5.1.

0 % False-Positive Rate on Validation Data

This method tunes the threshold on the *Mean Squared Error* (MSE) of the validation data. It is part of the data that simulates the nominal behavior of the test bench. The threshold is set to the maximum of the MSE which is 0.00068 for the selected model. An overview of the performance for the different fault types is presented in Table 4.5.

Table 4.5: Performance on different fault types for $0\,\%$ *False-Positive Rate on Validation Data, thus using a threshold of* 0.00068

Metric	All Faults	Sensor frozen	Sensor drift	Sensor offset	LOX leakage	Changed _CVO150
FPR	0.000	0.000	0.000	0.000	0.000	0.000
TPR	0.479	0.642	0.744	0.999	0.000	0.002
F1	0.648	0.782	0.853	0.999	0.000	0.004

In the case of the static threshold tuned on nominal data, the LOX leakage cannot be detected despite a relatively high rise in MSE, as depicted in Figure 4.3. However, for a frozen sensor, an anomaly is detected after a short delay, which is plotted in Figure 4.4. The MSE of this fault grows exponentially in the visible area. This indicates that the model is having more and more difficulties reconstructing these previously unseen measurements. The MSE for the LOX leakage stays below the selected threshold for the whole duration of the anomaly, as can be seen in Figure 4.3. A lower threshold would have detected almost the whole anomaly, even without FPs for this specific run. A similar argumentation is possible for the changed τ_{CVO150}, although with a lower threshold, more FPs, and longer periods without detection that are created by the nature of this anomaly. Such a threshold can be set with help of the ROCs.

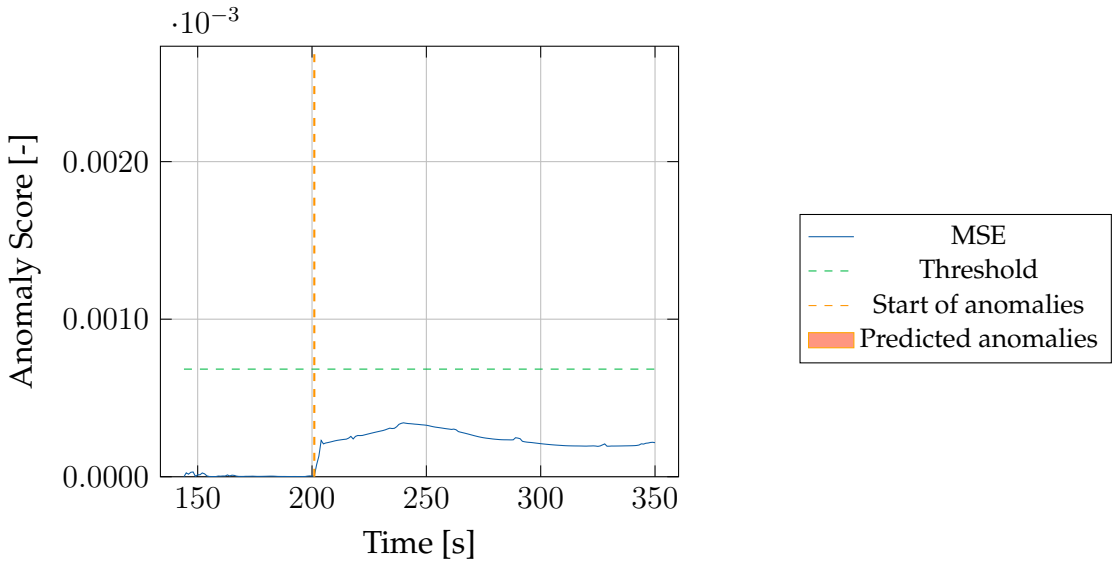


Figure 4.3: 1 % *False-Positive Rate on Validation Data* evaluated on a LOX leakage

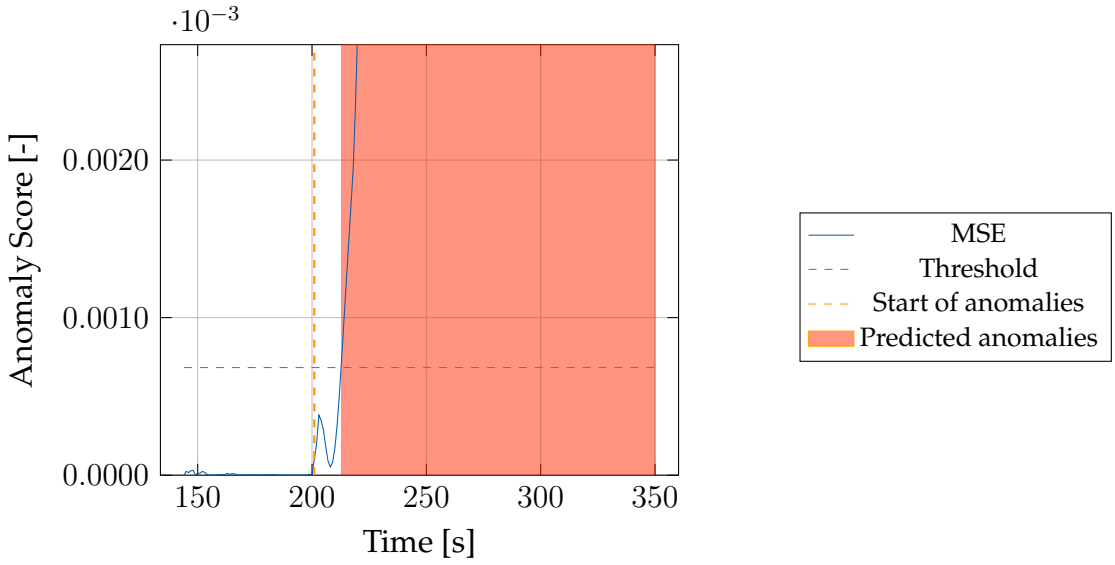


Figure 4.4: 1 % *False-Positive Rate on Validation Data* evaluated on a frozen sensor

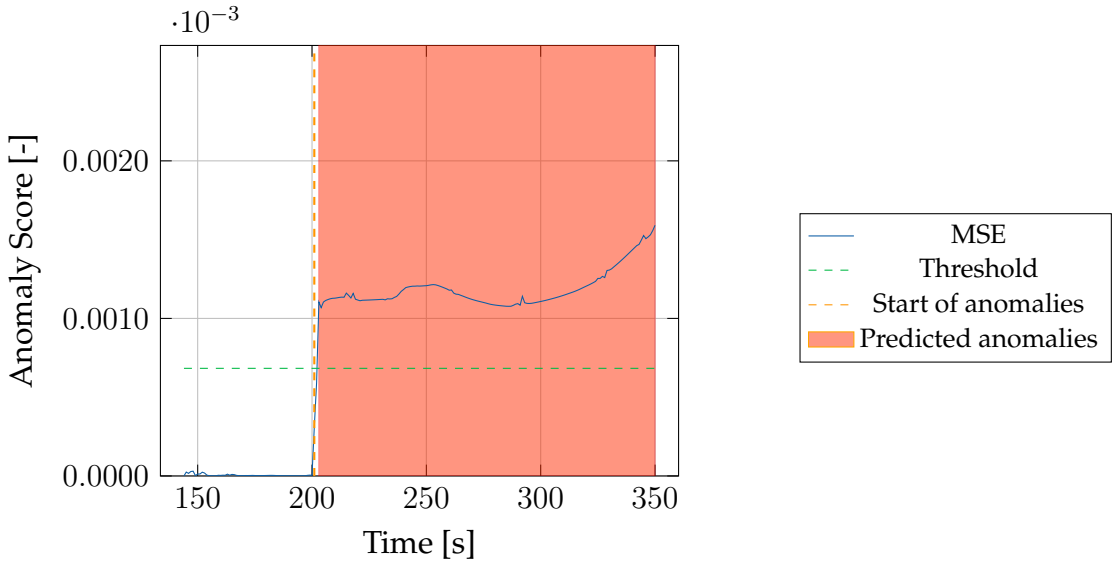


Figure 4.5: 1 % *False-Positive Rate on Validation Data* evaluated on a sensor offset

Receiver Operating Characteristics

For the labeled faulty data, the *Receiver Operating Characteristic*s (ROCs) curves are given for all fault types in Figure 4.6. In each plot, the location of the threshold set with FPR $= 1\,\%$, using all faulty data, is indicated by an orange dot. This corresponds to the requirement for AD in this thesis. The green diagonal line represents a hypothetical random-guessing classifier. The blue curve corresponds to the ROC of the evaluated model.

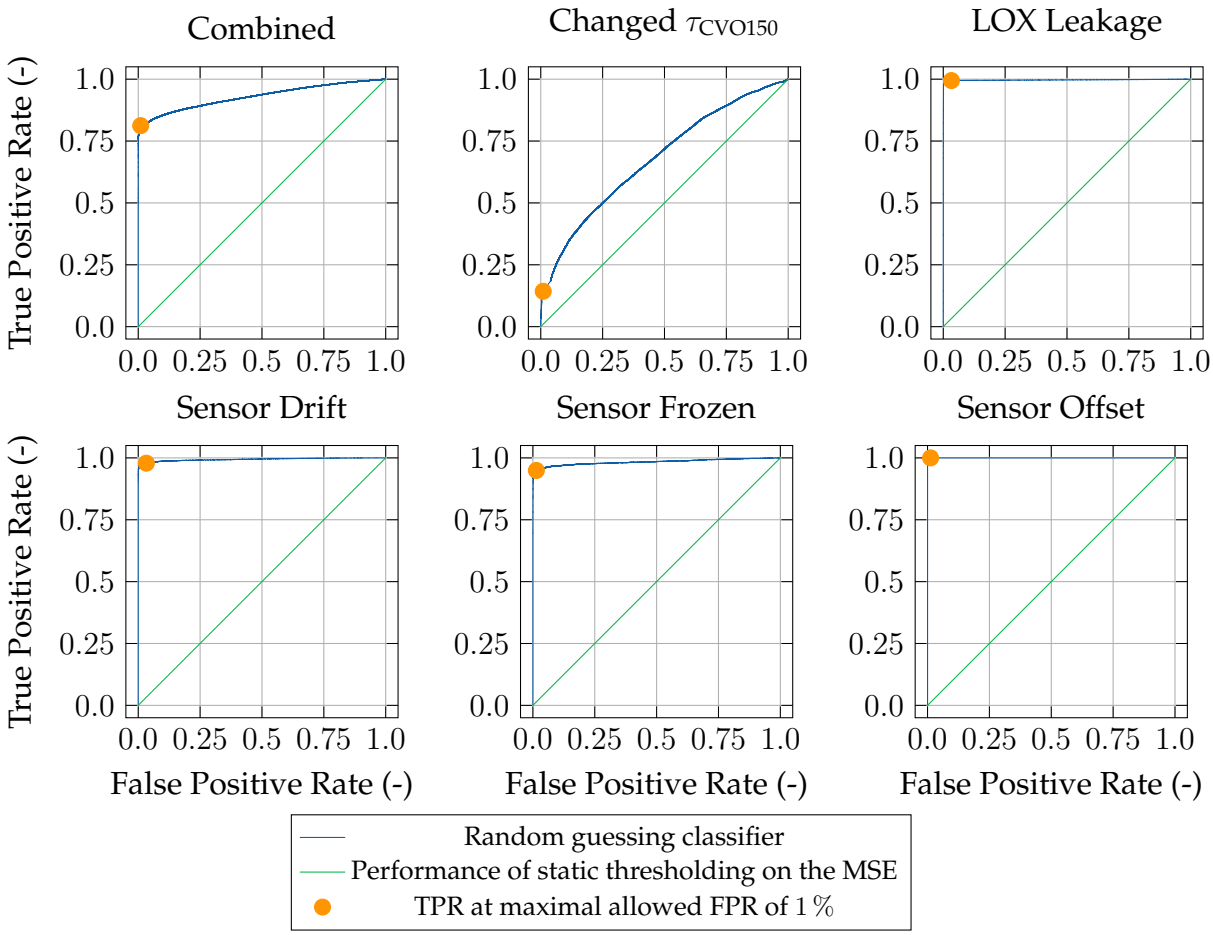


Figure 4.6: ROC curves for the selected fault types, the maximal allowed FPR is highlighted with an orange dot

For the LOX leakage and the sensor faults, the curves are close to an optimal classifier. This seems to contradict the results from the first presented thresholding method, but realistic ROC curves converge towards a TPR of 0 when no FPs are tolerated, when very large datasets are used, assuming *Gaussian noise*. However, the model seems to shows poor performance on the changed τ_{CVO150}. As the fault types are equally represented in the faulty dataset, the combined ROC curve corresponds to the average of the individual curves. Except for an ideal classifier when the integral of the ROC curve is 1.0, there

is always a trade-off between FPR and TPR. If the FPR should be kept low, this will result in a low *detection rate*. By allowing more *False Positive*s, more *True Positive*s will be detected, too.

$1\,\%$ False-Positive Rate on Faulty Data

While setting the threshold from nominal data is an unsupervised method, the nature of the faulty data can be used to tune the threshold. Based on the requirements, a tolerated *False Positive Rate* (FPR) of $1\,\%$ on the faulty data is used to tune the threshold. Table 4.6 presents an evaluation by fault type.

Table 4.6: Performance on different fault types for $1\,\%$ *False-Positive Rate on Faulty Data with a threshold value of* 1.34×10^{-5}

Metric	All Faults	Sensor frozen	Sensor drift	Sensor offset	LOX leakage	Changed τ_{CVO150}
FPR	0.010	0.010	0.006	0.009	0.013	0.012
TPR	0.790	0.916	0.961	1.000	0.994	0.080
F1	0.879	0.954	0.977	0.997	0.994	0.146

The frozen sensor in Figure 4.7 and the sensor offset in Figure 4.8 show that the lower threshold separates anomalies well from the nominal values at the starts of the runs. However, especially at the start of the run, a higher reconstruction error results in *False Positive*s. When the input features are analyzed, this higher MSE corresponds to a valve operation pattern. The system has a delay between sending commands to the valves, executing the commands, and the effect taking place. The spikes hint that the AE might have issues learning the temporal dependencies with the given *sliding windows*. A third example is presented with a changed τ_{CVO150}. A plot of this run can be found in Figure 4.9. For this fault type, anomalies are reported only for some short phases. As a τ_{CVO150} is the first order time constant of the `CVO150` valve, it can best be observed when the valve changes position. The position of the `CVO150` valve for the generic sequence together with the corresponding prediction is plotted in Figure 4.10. Except for the initial spikes, an anomaly is reported when the valve closes but not when it opens. The reason for this unexpected reconstruction behavior requires to understand which intrinsic features of the data have been learned by the FF AE model. However, the analysis of the model parameters lies definitely beyond the scope of this thesis.

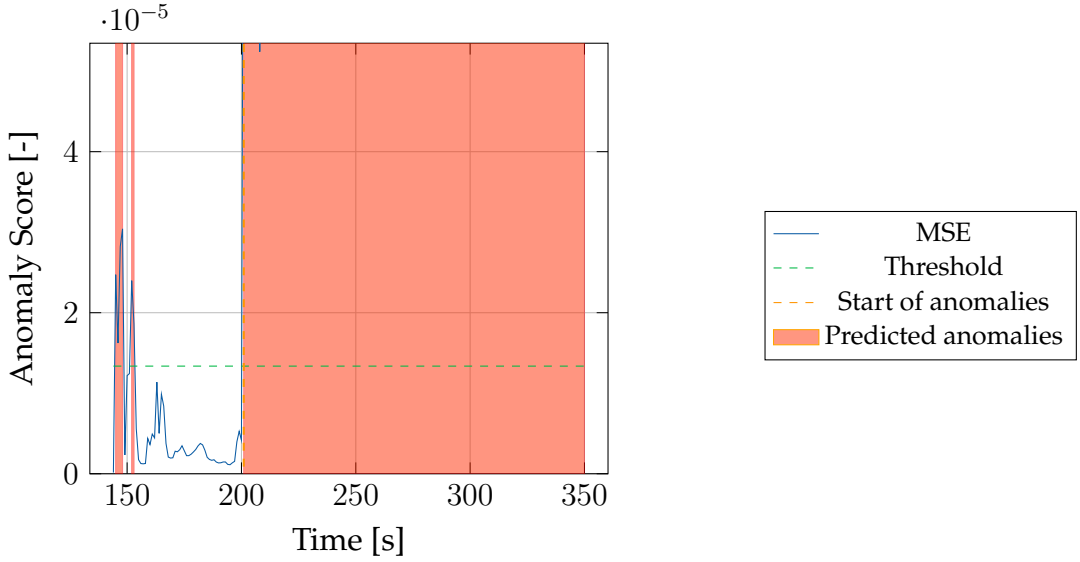


Figure 4.7: 1 % *False-Positive Rate on Faulty Data* evaluated on a frozen sensor

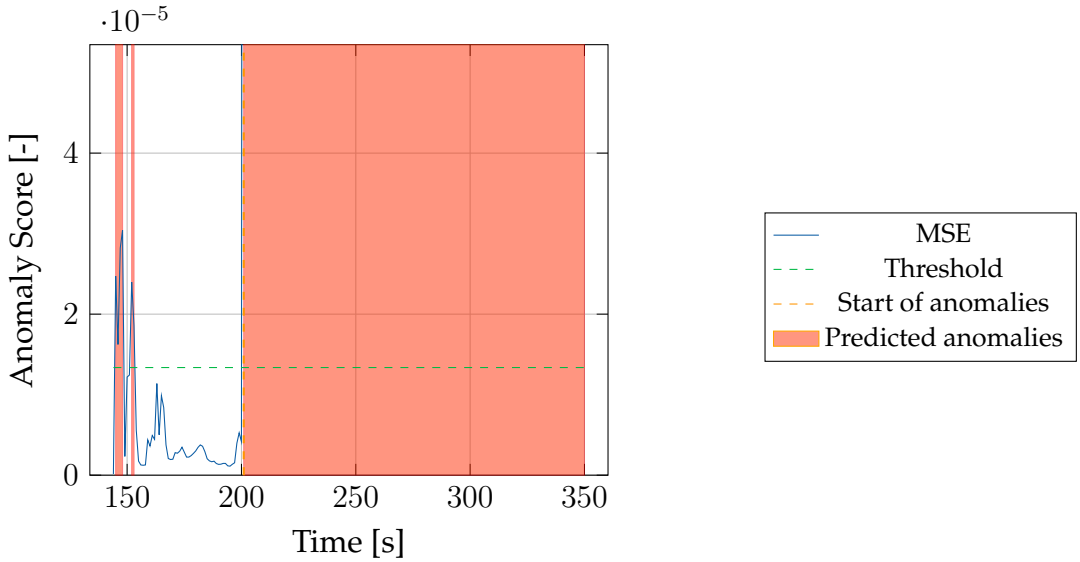


Figure 4.8: 1 % *False-Positive Rate on Faulty Data* evaluated on a sensor offset

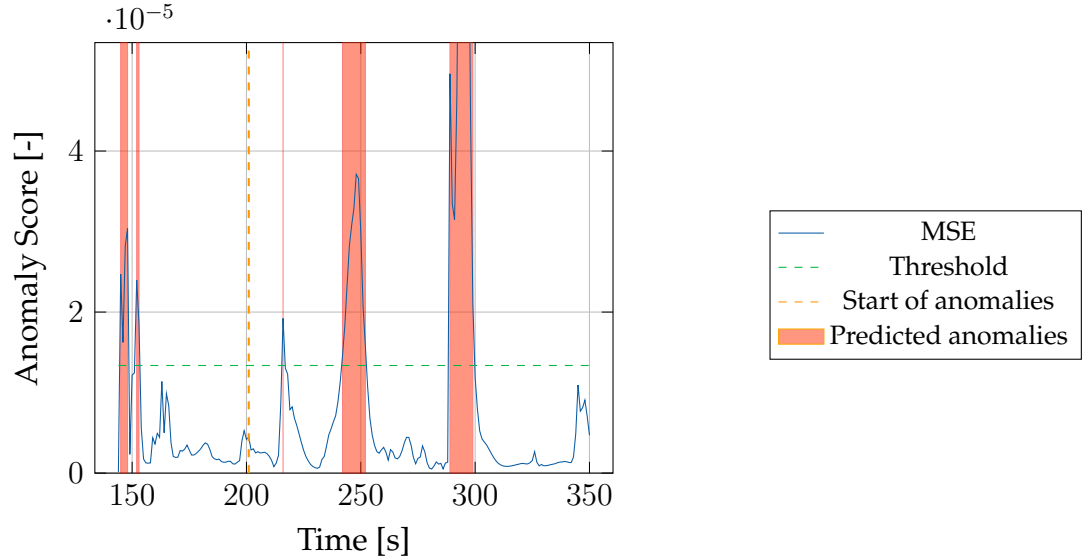


Figure 4.9: 1 % *False-Positive Rate on Faulty Data* evaluated on a changed τ_{CVO150}

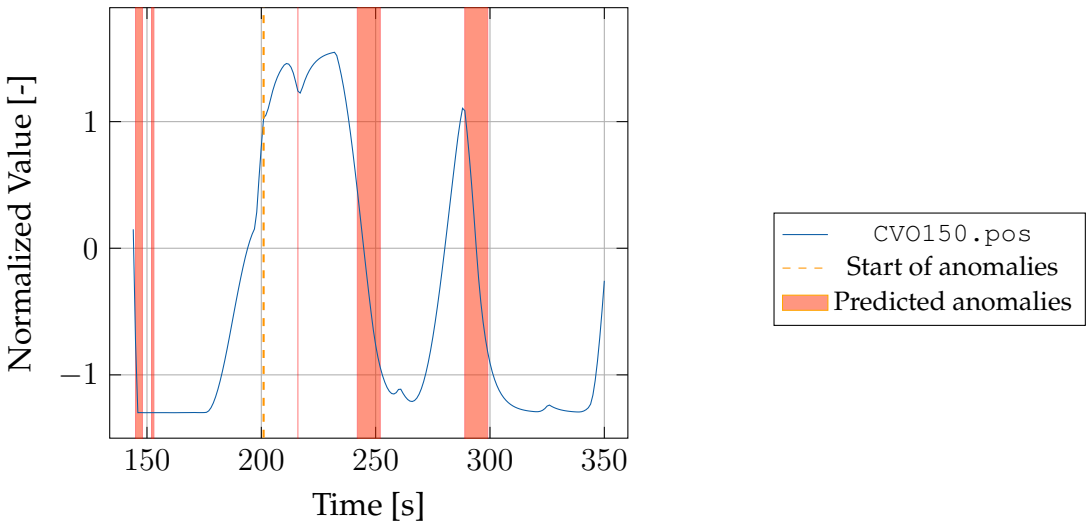


Figure 4.10: The `CVO150` valve for the generic sequence, the corresponding anomaly prediction is highlighted in red

4.4.2 Modifications of the Static Threshold

The *static thresholding* can be tuned to reduce the number of false positives. However, this also leads to a reduction in *True Positive*s. The methods presented in the following aim to identify structures in the detection results that point to possible *False Negative*s. This works best if the structure of the anomalies is taken into account, as described in Chapter 3.1.5. The proposed methods require a delay of a few seconds as they include future reconstruction errors or predicted labels in order to identify whether an anomaly should be kept or dropped.

Remove Spikes

Assuming that anomalies have a minimal length of t, short spikes of a few seconds should be ignored. Thus, any sequence of positive labels in the prediction that is shorter than t can be reclassified to nominal. If this method should be used to increase the TPR too, the following approach can be used:

Algorithm 1 RemoveSpikes

Let A be the reconstruction error list
Let a be a value slightly less than the threshold that should be used later.
Let b have a value slightly less than a.
Let t be the minimal duration of anomalies.
Let $l = 0$ be the length of the current anomaly
for $i = 1, 2, \ldots$ **do**
    if $A_{\mathrm{i}} > a$ **then**
        increment l
    else if $A_{\mathrm{i}} \leq a$ and $l > 0$ **then**
        set $A_{\mathrm{i-l}}$, $A_{\mathrm{i-l+1}}$, …, A_{i} to b
        set $l = 0$
    else
        set $l = 0$

In the following, a minimal duration of anomalies of $t = 10$ is assumed. Depending on the selected threshold, this method either increases the TPR by about one percent or decreases the FPR by a factor of seven while decreasing the TPR slightly due to pruned short spikes in the anomalous regions. For the latter variant, the metrics for the different fault types are shown in Table 4.7.

The MSE plots illustrate how the spikes from the start of the sequence are mitigated by reducing the MSE from the original result. For the generic sequence, no FP is left in the nominal part. In Figure 4.11, the two longer positive predictions are kept while a short spike is mitigated. For a purely nominal run, as depicted in Figure 4.12, no fault is detected. The frozen sensor in Figure 4.13 can be clearly separated from the nominal part of the run.

Table 4.7: Performance on different fault types for *Remove Spikes*, using 1.27×10^{-5} as threshold

Metric	All Faults	Sensor frozen	Sensor drift	Sensor offset	LOX leakage	Changed τ_{CVO150}
FPR	0.001	0.001	0.001	0.001	0.002	0.002
TPR	0.783	0.915	0.961	1.000	0.994	0.046
F1	0.878	0.956	0.961	1.000	0.994	0.088

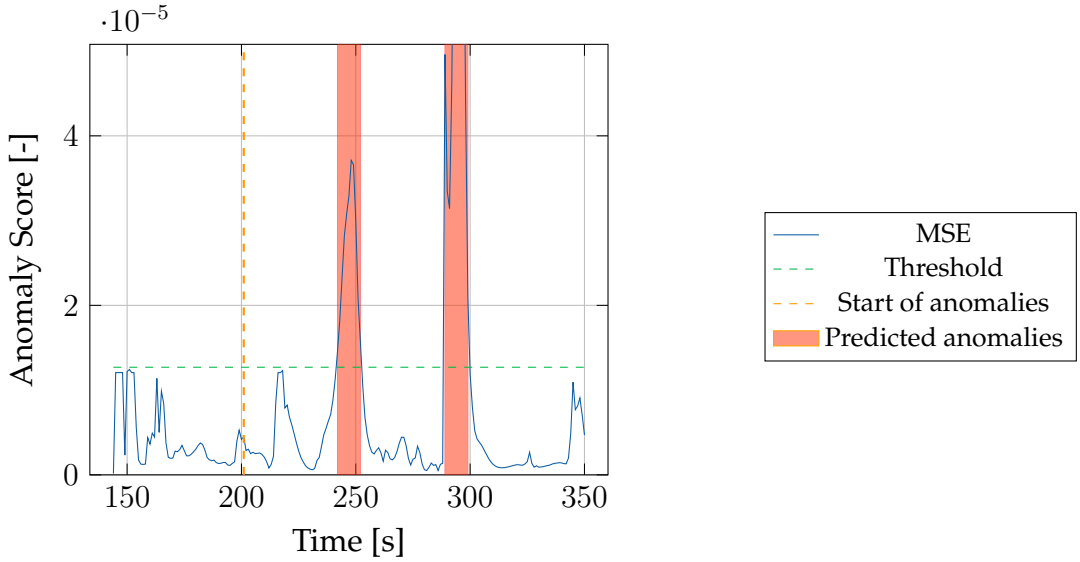


Figure 4.11: *Remove Spikes* evaluated on a changed τ_{CVO150}

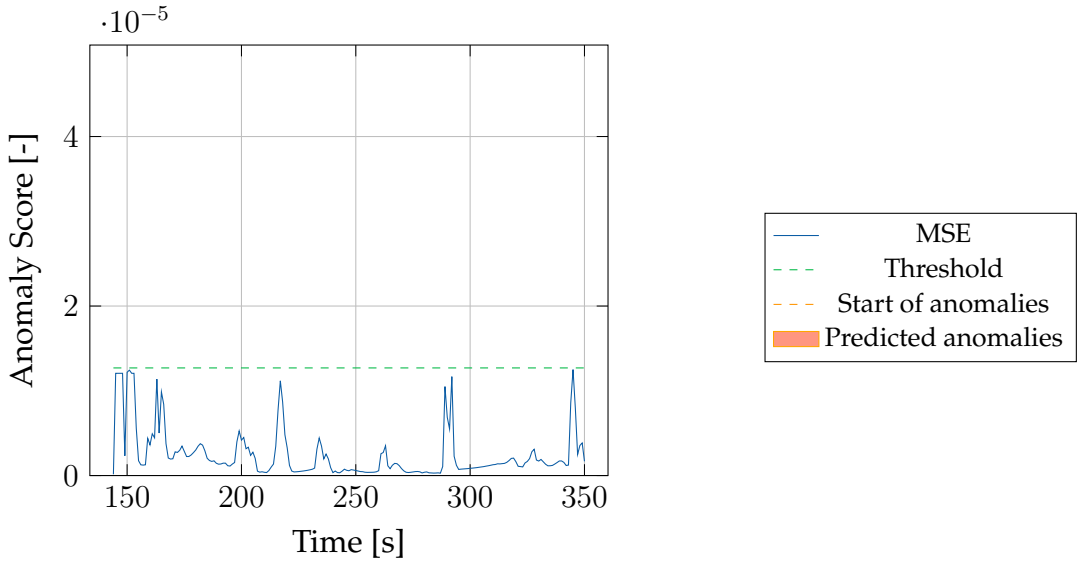


Figure 4.12: *Remove Spikes* evaluated on a nominal run

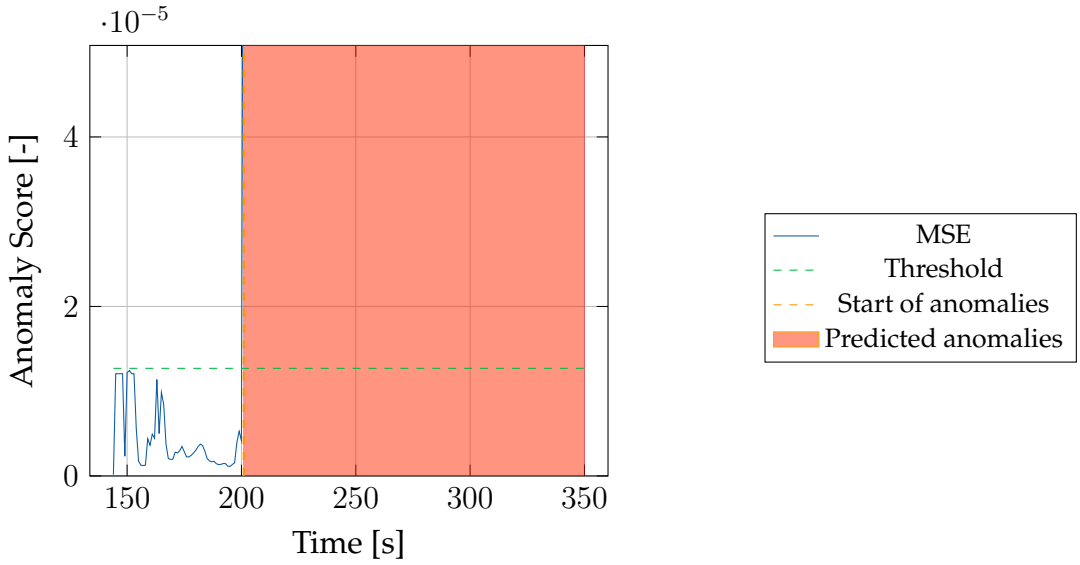


Figure 4.13: *Remove Spikes* evaluated on a frozen sensor

Flag Faulty

This method makes use of the special structure of the faulty test runs as described in Section 3.1.5. As soon as a fault is introduced, it stays until the end of a run. That means, if there is a high probability for an anomaly, the following values can be flagged as anomalous even if they do not qualify as anomalous from the AD. However, it is important to note that in case of any FP, many nominal values will be flagged as anomalous. This can lead to a high and thus undesired increase of the FPR. In order to avoid flagging a fault too early, the spikes are first removed before the faults are flagged. In the implementation, the *reconstruction error* is modified. As soon as a value surpasses the anomaly threshold, the following values are set to a higher value. This higher value is defined to be the maximum of all previous values to keep the structure of the *reconstruction error* to some extent for manual analysis. Otherwise, it could also be set to a high constant.

The metrics for each fault type are shown in Tables 4.8. The thresholds have been tuned for a *False Positive Rate* of $1\,\%$. As the changed τ_{CVO150} is nearly invisible when the `CVO150` valve is kept at a position, flagging should have the greatest impact for this type of fault. The corresponding test run is shown in Figure 4.14.

Table 4.8: Performance on different fault types for *Flag Faulty* with a threshold of 1.27×10^{-5}

Metric	All Faults	Sensor frozen	Sensor drift	Sensor offset	LOX leakage	Changed τ_{CVO150}
FPR	0.010	0.011	0.000	0.016	0.014	0.008
TPR	0.824	0.936	0.948	1.000	0.993	0.246
F1	0.901	0.964	0.948	0.995	0.992	0.393

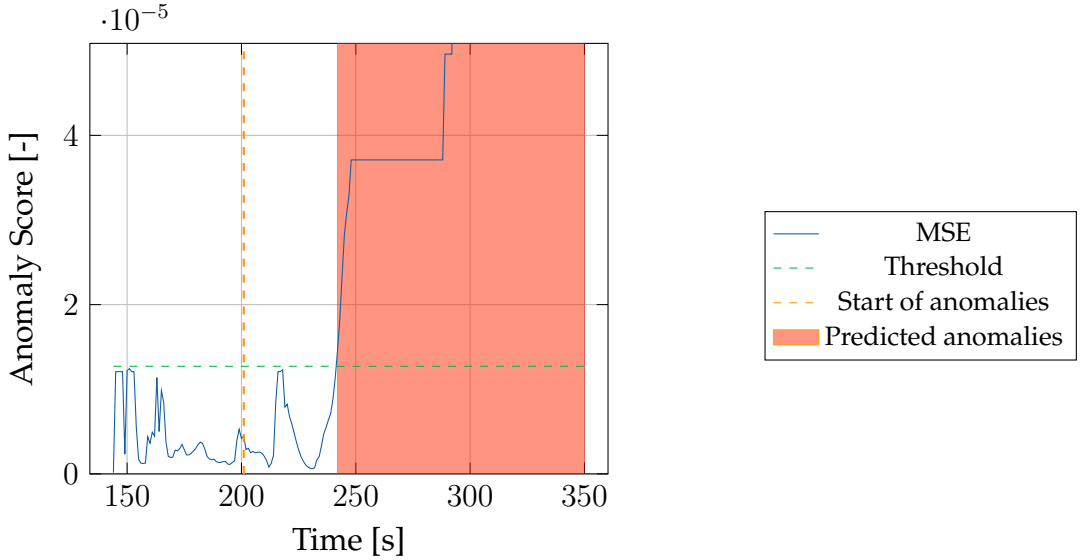


Figure 4.14: *Flag Faulty* on a changed τ_{CVO150}

4.4.3 Other Anomaly Detection Methods

While a static threshold and its variations work well in static environments, they suffer under dynamic and more realistic conditions. However, as this work is using synthetic data under clearly defined conditions, the main motivation for the test of other AD methods is the following observation: When comparing Figure 4.12, which is the run without any faults, with Figure 4.11, which depicts the changed τ_{CVO150}, the MSE is visibly higher when the fault is active compared to the nominal version. If a method could use the context and detect an anomaly if the MSE behaves in a suspicious way, this could increase the detection rate and maybe even mitigate some more *False Positive*s.

Parametric thresholding

This method calculates an anomaly likelihood based on the mean and variance of a long window and the mean of a short window as described in Section 3.5.2. Here, the long window is set to 60 prior values, while the short window consists of ten prior values. The value for the long window is actually limited by the minimal run length in the faulty data. In order to be able to detect anomalies close to the start of a run, the mean and variance of the long window before the 60^{th} value are calculated taking the weighted average of a shorter window and the mean and variance from 2100 values sampled from the nominal part of the faulty runs. As this method cannot be tuned from the ROC but only with an *anomaly parameter* ϵ, the FPR for $\epsilon = 0.01$ is very high, as presented in Table 4.9. Unfortunately, the vast majority of anomaly likelihoods are either 0 or 1 so that the ϵ parameter cannot be fine-tuned well.

Table 4.9: Performance on different fault types for *Parametric Thresholding*

Metric	All Faults	Sensor frozen	Sensor drift	Sensor offset	LOX leakage	Changed τ_{CVO150}
FPR	0.293	0.283	0.299	0.295	0.287	0.293
TPR	0.841	0.832	0.984	1.000	0.996	0.391
F1	0.828	0.829	0.904	0.913	0.912	0.496

The results of this method evaluated on a changed τ_{CVO150} and the nominal version are shown in Figures 4.15 and 4.16. Unfortunately, this method detects a high amount of *False Positive*s for the runs, while the detection of the changed τ_{CVO150} is also limited. To understand this behavior, the effects of the two window sizes need to be described. The short sliding window smooths the MSE by calculating an average. The long sliding window however provides a context with long-term nominal behavior. In order not to skip the first 60 values of each run, the corresponding long windows have been combined with another nominal sample as described in Section 3.5.2. As the nominal MSE of the faulty runs is on average much lower than the start of the generic sequence, anomalies are detected right at the beginning. The same problem occurs in later parts

of the nominal run. The sliding window of length 60 adapts to quickly to low values so that even the relatively low spikes in the later part are predicted to be anomalous.

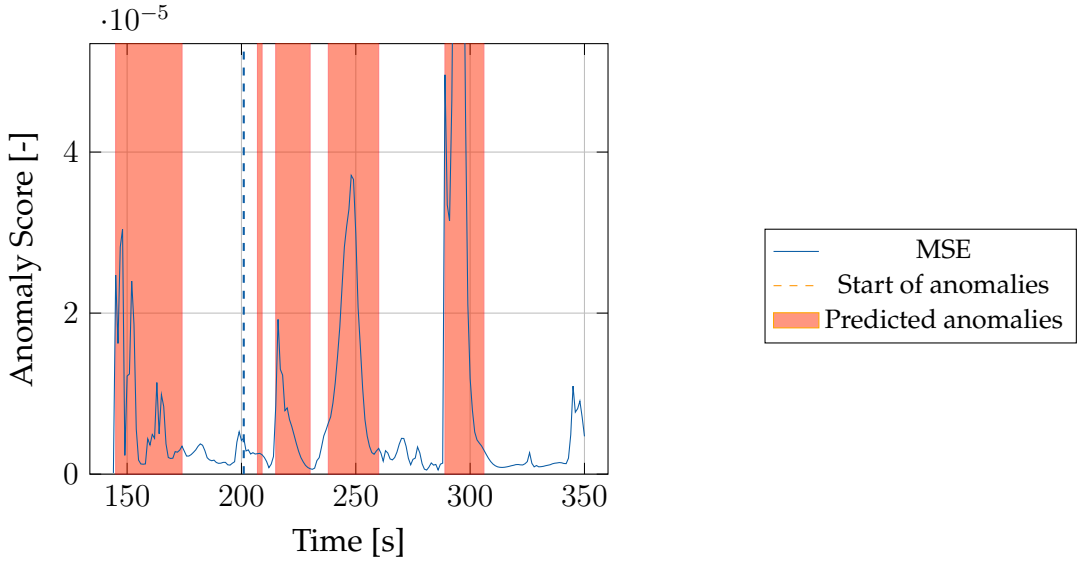


Figure 4.15: *Parametric Thresholding* evaluated on a changed τ_{CVO150}

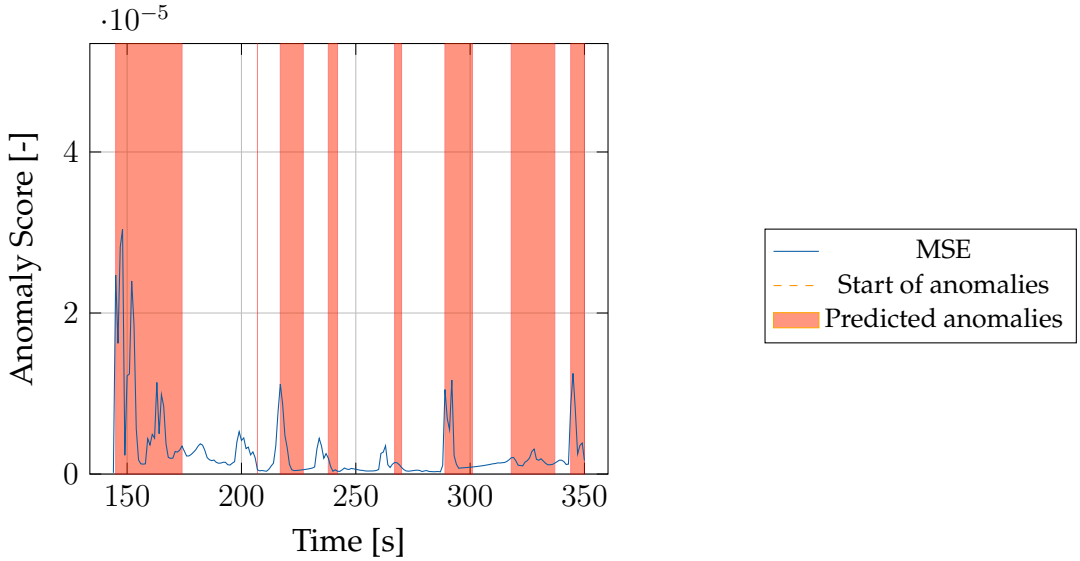


Figure 4.16: *Parametric Thresholding* evaluated on a nominal run

Peaks-Over-Thresholds

Due to the poor performance of the *Parametric Thresholding*, another unsupervised, statistical method is investigated. The *Peaks-Over-Threshold* (POT) method is evaluated in its streaming variant with drift. A detailed description of this method can be found in Section 3.5.2. While the initialization uses the nominal part of the faulty runs, the other

parameters remain unchanged from the version of Siffer *et al.* [13]. Thus, the probability for anomalies is set to 10^{-3}, the depth to 450 [1], and the level parameter to 0.98. Tuning these values to an optimum would require a search similar to the HP study as three-variable optimization problems result in too many possible combinations to be tuned manually which is not in the scope of this thesis. The individual runs are analyzed separately as they are independent of each other. The results of each run are then concatenated for further analysis. The metrics for each fault type can be found in Table 4.10. In comparison, the static thresholding achieves an overall TPR of 0.752 for the same FPR.

Table 4.10: Performance on different fault types for POT

Metric	All Faults	Sensor frozen	Sensor drift	Sensor offset	LOX leakage	Changed τ_{CVO150}
FPR	0.001	0.001	0.001	0.001	0.001	0.001
TPR	0.751	0.849	0.924	1.000	0.958	0.024
F1	0.857	0.918	0.960	1.000	0.978	0.047

The changed τ_{CVO150} in Figure 4.17, the LOX leakage in Figure 4.18, and the sensor drift in Figure 4.19 demonstrate that the spikes at the beginning are not detected by the method. The actual anomalies are detected, with a huge delay for the changed τ_{CVO150} and a shorter delay for the sensor drift.

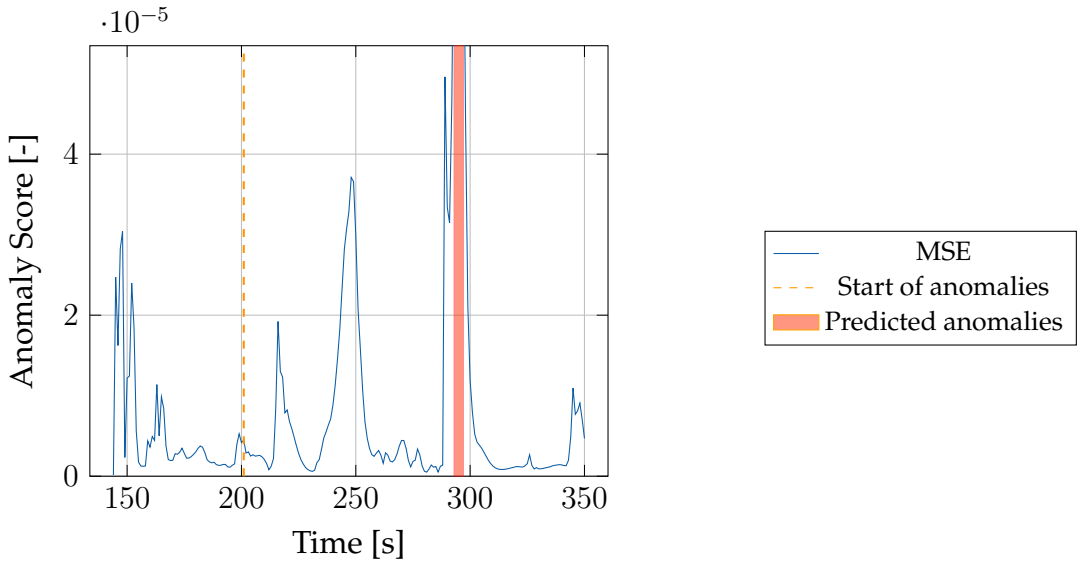


Figure 4.17: drifting, streaming POT evaluated on a changed τ_{CVO150}

[1] Which is longer than many run lengths but should be handled by the method

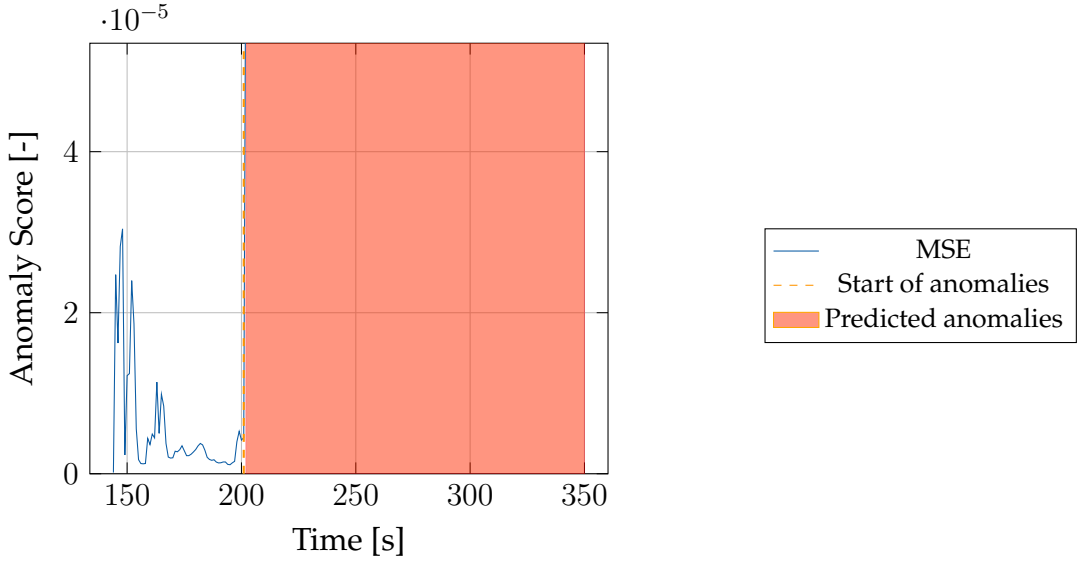


Figure 4.18: drifting, streaming POT evaluated on a LOX leakage

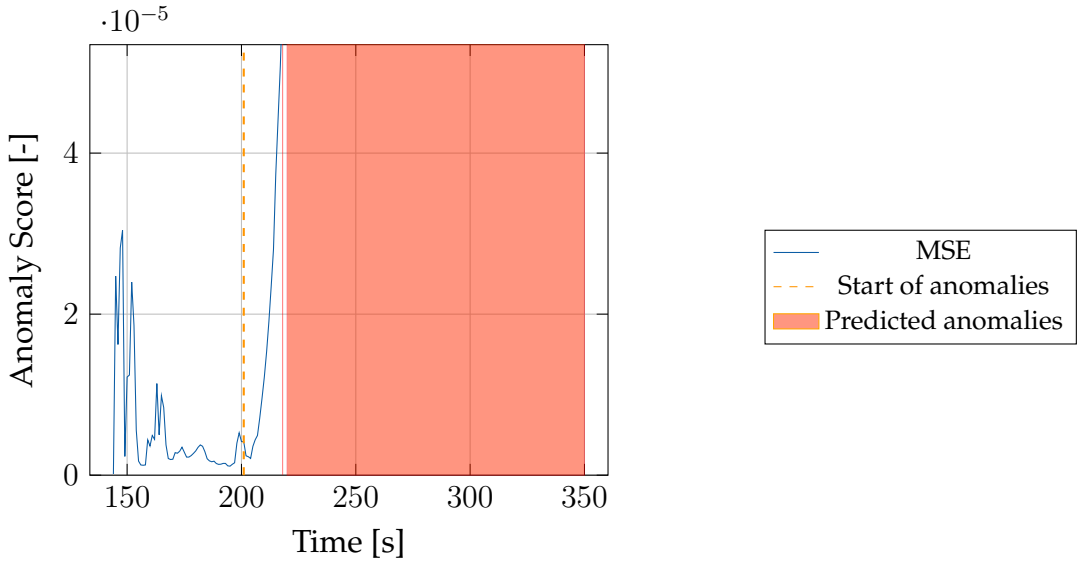


Figure 4.19: drifting, streaming POT evaluated on a sensor drift

4.5 Comparison

This section uses *F1 score*, *precision* and *recall* to compare the presented methods for AD.

Table 4.11: Scores for all analyzed AD methods on the whole faulty dataset

	0 % FPR on val. data	1 % FPR on faulty	Remove Spikes	Flag faulty	Parametric thresholding	POT
FPR	**0.000**	0.010	0.001	0.010	0.293	0.001
F1	0.65	0.88	0.88	**0.90**	0.83	0.86
Precision	**1.000**	0.992	0.999	0.992	0.817	0.999
Recall	0.48	0.79	0.78	0.83	**0.84**	0.75

The scores, together with the overall FPR, are listed in Table 4.11. *0 % False-Positive Rate on Validation Data* has the highest precision and the lowest FPR, F1 and recall. *Parametric Thresholding* has the highest FPR, the lowest precision and the highest recall. Both are heavily imbalanced, which means they do not achieve the best possible detection out of the given reconstruction error from the AE model.

Out of the four remaining methods, the *Flag Faulty* method shows the best metrics except for the FPR which is lower for POT and for *Remove Spikes*. The other methods *Parametric Thresholding* and *Peaks-Over-Threshold* underperform against the methods based on a static threshold. As the dataset is synthetic and does not even contain artificial noise, these methods cannot demonstrate their full potential. With the analysis of individual runs, it becomes clear that none of the proposed methods can effectively utilize the patterns in the reconstruction error to detect significantly more anomalous patterns. However, removing spikes and flagging faults until the end of the runs demonstrate that handcrafted methods can indeed reduce the amount of FPs while even increasing the number of TPs.

For the different fault categories, most methods perform well on the sensor faults and the detection of the LOX leakage. However, the changed τ_{CVO150} cannot be detected well for any of the methods. At first glance, this seems to be an issue to be solved by using another AE model that results in higher reconstruction errors for the whole duration of the changed τ_{CVO150}. However, the description of the fault suggests that the fault is only visible when the `CVO150` valve is moved. With information gained from the analysis of a generic fault, the FF AE produces a high reconstruction error when the `CVO150` valve is closed. To answer why the opening of the valve can be reconstructed properly, an analysis of the model parameters would be required.

Chapter 5

Summary and Conclusion

In this thesis, *Machine Learning* (ML) based *Anomaly Detection* (AD) has been evaluated using *Feedforward Auto-Encoder*s (FF AEs) and a collection of semi-unsupervised thresholding methods and statistical methods to predict anomalies.

In the operation of rocket engine test bench systems, it is crucial to detect anomalies to prevent major destruction in the event of failure. Significant anomalies causing strong deviations from the expected values can be detected by applying fixed thresholds to selected sensor values. The run is aborted if this health monitoring system records potentially dangerous values. Following these rules strictly can result in unnecessary test aborts as not every extreme sensor reading indicates a major malfunction in the test bench. Accordingly, a system is designed with the help of ML in order to detect anomalies with higher reliability.

An FF AE is trained on purely nominal data that is generated from a simulation of the test bench. In this setup, the FF AE will learn to reconstruct nominal data well, while any anomaly should result in a high reconstruction error. The tuning and evaluation of the AD methods takes place on labeled test data containing different types of anomalies. While the methods of this thesis are evaluated on a proprietary, synthetic dataset from a rocket engine test facility, the insights gained from working on this data can be transferred to datasets with similar characteristics.

A static threshold and variations thereof to increase its performance on the used dataset have been evaluated against statistical methods. The statistical methods include a form of *parametric thresholding* as proposed by Hundman *et al.* [1] together with the *Peaks-Over-Threshold* (POT) approach from Siffer *et al.* [13]. While *parametric thresholding* suffers from numerical issues and a high *False Positive Rate* (FPR), POT reaches a performance similar to static thresholding, albeit further tuning is required to demonstrate the full potential of this method. Setting the threshold high enough such that no anomaly would be detected in the validation dataset proves inefficient with an F1 score of 0.65 while allowing up to $1\,\%$ *False Positive*s (FPs) in the faulty dataset results in a much

higher F1 score of 0.88. While a removal of short *spikes* in the reconstruction error does not change the performance significantly, further utilization of the structure of the anomalies results in another improvement in FPR, F1 and recall with a constant precision. This method keeps anomalies flagged until the end of each run, resulting in an F1 score of 0.90.

ML-based AD combines methods from two different fields. While the training of *Neural Network*s (NNs) is based on heuristics and best-practices, AD relies on statistical methods. While the *Auto-Encoder* (AE) and the actual AD are two distinct parts of the method, they have been integrated and adapted to each other. At the same time, the inherent modularity has been used to test replacements for the AD component. The same could be done for the AE part of the method.

5.1 Lessons Learned

Unfortunately, the implemented methods are not yet reliable enough to support the *redline* approach (applying absolute limits on selected sensor readings) currently used in the test facility. The detection results should be taken with a grain of salt by the operators. The limited size of the nominal rocket engine test facility (*P5 Liquid Oxygen* (LOX), described in Section 3.1) dataset and the structure of its test dataset are challenges for the successful implementation of AD. Strikingly, a purely nominal dataset is the best way to make sure that an AE will struggle to reconstruct any kind of anomaly. And at the same time, the dataset imposes difficulties on the implementation with its special structure of many independent short time series.

Time series datasets often contain temporal dependencies. Using just the current values of the features as input for an FF AE will not be able to properly reconstruct any features that are connected by temporal dependencies. While *sliding windows* can be used as a workaround to use FF AEs on time series data, they do not perform much better on the given dataset compared to FF AEs without *sliding windows* . In fact, a *Hyper-Parameter* (HP) study on the *sliding window size* produces way shorter window sizes than anticipated.

Another surprising result of the HP study is the high performance of *overexpanding* models compared to their *bottlenecked* counterparts. As observed by Yong and Brintrup [23], these architectures should be taken into account for the ML model used for reconstruction-based AD.

5.2 Future Work

As several variants of AD have been researched by this thesis, the insights from these methods make the quest for future work. While this work demonstrates the potential of reconstruction-based AD itself, the AE component has never been changed in principle.

Instead of the commonly used FF AE which is not designed for time series data, the performance of *Long Short-Term Memory* (LSTM) AEs should be investigated. However, they require substantially more computation time and memory for training and prediction than FF AEs, an evaluation of this type of AE cannot be part of this thesis due to a lack of computational resources.

As the data is synthetic, adaption to more realistic scenarios is necessary. While all imaginable types of faults in the system and the sensors should be generated, this will also include the addition of artificial noise and a fine-tuning of the AE and the AD methods on real data. It is questionable whether the best combination of this work could handle this scenario based on the assumptions necessary to compute a static threshold. Instead of the used FF AE, a *variational* AE might be useful. In addition, the statistical methods might finally outperform the methods based on a static threshold. In general, other methods for *univariate* time series should be applied to the reconstruction error.

For the proposed application itself, other non-ML based AD methods should be evaluated. It is possible that statistical methods applied to the features can work better than the current approach of *redlines* while reducing the number of FPs. They might as well give a third opinion on the likelihood of a fault in the system.

Bibliography

[1] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding", *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 387–395, 2018. DOI: 10.1145/3219819.3219845.

[2] J. Blumenfeld, *SpaceML: Rise of the Machine (Learning)*, https://www.earthdata.nasa.gov/learn/articles/spaceml, accessed May 30, 2023, May 2021.

[3] K. Dresia, E. Kurudzija, G. Waxenegger-Wilfing, *et al.*, "Automation of Testing and Fault Detection for Rocket Engine Test Facilities with Machine Learning", *International Journal of Energetic Materials and Chemical Propulsion*, 2023, under review.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey", *ACM Comput. Surv.*, vol. 41, no. 3, 2009. DOI: 10.1145/1541880.1541882.

[5] M. E. Tschuchnig and M. Gadermayr, "Anomaly detection in medical imaging - a mini review", in *Data Science Analytics and Applications*, Springer Fachmedien Wiesbaden, 2022, pp. 33–38. DOI: 10.1007/978-3-658-36295-9_5.

[6] D. A. Bierbrauer, A. Chang, W. Kritzer, and N. D. Bastian, "Cybersecurity anomaly detection in adversarial environments", *arXiv*, 2021. DOI: https://doi.org/10.48550/arXiv.2105.06742.

[7] P. Tiwari, S. Mehta, N. Sakhuja, J. Kumar, and A. K. Singh, "Credit card fraud detection using machine learning techniques: A comparative analysis", in *2017 International Conference on Computing Networking and Informatics (ICCNI)*, 2017, pp. 1–9. DOI: 10.1109/ICCNI.2017.8123782.

[8] M. Óskarsdóttir, W. Ahmed, K. Antonio, B. Baesens, R. Dendievel, T. Donas, and T. Reynkens, "Social network analytics for supervised fraud detection in insurance", *Risk analysis : an official publication of the Society for Risk Analysis*, vol. 42, 2020. DOI: 10.1111/risa.13693.

[9] A. Chriki, H. Touati, H. Snoussi, and F. Kamoun, "UAV-based surveillance system: An anomaly detection approach", in *2020 IEEE Symposium on Computers and Communications (ISCC)*, Rennes, France, 2020, pp. 1–6. DOI: 10.1109/ISCC50000.2020.9219585.

[10] F. Y. Edgeworth, "Xli. on discordant observations", *Philosophical Magazine Series 1*, vol. 23, pp. 364–375, 1887. DOI: 10.1080/14786448708628471.

[11] V. Hodge and J. Austin, "A survey of outlier detection methodologies", *Artificial Intelligence Review*, vol. 22, pp. 85–126, 2004. DOI: 10.1023/B:AIRE.0000045502.10941.a9.

[12] M. ElDali and K. D. Kumar, "Fault diagnosis and prognosis of aerospace systems using growing recurrent neural networks and LSTM", in *2021 IEEE Aerospace Conference (50100)*, virtual, 2021, pp. 1–20. DOI: 10.1109/AERO50100.2021.9438432.

[13] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet, "Anomaly Detection in Streams with Extreme Value Theory", in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17, Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 1067–1075. DOI: 10.1145/3097983.3098144.

[14] F. Wang, K. Wang, and B. Yao, "Time series anomaly detection with reconstruction-based state-space models", *arXiv*, 2023. DOI: https://doi.org/10.48550/arXiv.2303.03324.

[15] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network", in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19, Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2828–2837. DOI: 10.1145/3292500.3330672.

[16] I. T. Jolliffe and J. Cadima, "Principal component analysis: A review and recent developments", *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 374, no. 2065, pp. 201–202, 2016. DOI: 10.1098/rsta.2015.0202.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016, http://www.deeplearningbook.org.

[18] I. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions.", *SN COMPUT. SCI.*, vol. 2, no. 420, pp. 9–10, 2021. DOI: 10.1007/s42979-021-00815-1.

[19] T. G. Robertazzi and L. Shi, "Machine learning in networking", in *Networking and Computation: Technology, Modeling and Performance*. Cham: Springer International Publishing, 2020, pp. 151–190. DOI: 10.1007/978-3-030-36704-6_7.

[20] *Ecosim espss product overview*, https://www.ecosimpro.com/products/espss/, Accessed: 2023-05-17, 2016.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in python", *Journal of machine learning research*, vol. 12, pp. 2825–2830, 2011. DOI: https://doi.org/10.48550/arXiv.1201.0490.

[22] J. Kukaka, V. Golkov, and D. Cremers, "Regularization for deep learning: A taxonomy", *ArXiv*, 2017. DOI: https://doi.org/10.48550/arXiv.1710.10686.

[23] B. X. Yong and A. Brintrup, "Do Autoencoders Need a Bottleneck for Anomaly Detection?", *IEEE Access*, vol. 10, pp. 78 455–78 471, 2022. DOI: 10.1109/ACCESS.2022.3192134.

[24] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark", *Neurocomputing*, 2022. DOI: http://dx.doi.org/10.1016/j.neucom.2022.06.111.

[25] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice", *Neurocomputing*, vol. 415, pp. 295–316, 2020. DOI: 10.1016/j.neucom.2020.07.061.

[26] T. Gneiting and P. Vogel, "Receiver operating characteristic (ROC) curves", *arXiv*, 2018. DOI: https://doi.org/10.48550/arXiv.1809.04808.

[27] M. Viswanathan. "Q function and Error functions : demystified". (2012), [Online]. Available: https://www.gaussianwaves.com/2012/07/q-function-and-error-functions/.

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

[29] W. T. Falcon. "Pytorch lightning". (Mar. 2019), [Online]. Available: https://www.pytorchlightning.ai.

[30] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy", *Nature*, vol. 585, no. 7825, pp. 357–362, 2020. DOI: 10.1038/s41586-020-2649-2.

[31] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework", in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. DOI: 10.1145/3292500.3330701.

A Appendix

A.1 Frameworks

The frameworks in this section are used to implement the concepts of this thesis.

Pytorch Lightning

> *Pytorch Lightning* is the deep learning framework with batteries included for professional AI researchers and machine learning engineers who need maximal flexibility while super-charging performance at scale. [29]

Pytorch Lightning (PL) ships with the `LightningModule` to be used for the model to be trained, the `Trainer` wrapping all the training process and to deal with datasets, the `LightningDataModule`. Pure Pytorch code can easily be transformed to make use of the framework. PL shows its full potential when training on *Graphics Processing Unit*s (GPUs) comes into play. Lots of common mistakes and bugs can be avoided by not reinventing the wheel each time a new *Machine Learning* (ML) setup is being created. This reduces the time required to implement a model.

Scikit-learn

> Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and *Application Programming Interface* (API) consistency. [21]

While *Scikit-learn* also offers *Central Processing Unit* (CPU)-based approach to models and model training, the plenty of surrounding algorithms present a valuable source for pre-processing, post-processing and analysis of data. In this thesis, *Scikit-learn* 1.2.0 is used for scaling, the *Principal Component Analysis* (PCA), and the calculation of scores and the *Receiver Operating Characteristic*s (ROCs).

Numpy

> Array programming provides a powerful, compact and expressive syntax for accessing, manipulating and operating on data in vectors, matrices and higher-dimensional arrays. NumPy is the primary array programming library for the Python language. It has an essential role in research analysis pipelines in fields as diverse as physics, chemistry, astronomy, geoscience, biology, psychology, materials science, engineering, finance and economics. [30]

Numpy provides a solid basis for efficient computations using *Python*. For *Scikit-learn*, the *Numpy* `array` represents the data container. It can be converted to the *Torch* `Tensor` representation and the `DataFrame` form of *Pandas* easily.

Pandas

Pandas is a Python framework for tabular data. It is used to read and write tabular data and select columns thereof.

Optuna

> Optuna [is] an optimization software [...]. As an optimization software designed with define-by-run principle, Optuna is particularly the first of its kind. [31]

Optuna allows to set up a search space for several parameters easily and run a study to find an optimal value. This is combined with (optional) *pruning* of non-promising *trials* as well as different sampling methods in order to define the *trials*. *Optuna* can be used with any *Python* ML framework and only requires few new lines of code.

A.2 Figures

The figures in this section demonstrate the influence of anomalies on the feature reconstructions by the *Auto-Encoder* (AE). Plots without visible differences between original and reconstruction have been dropped.

τ_{CVO150}

The influence of this fault type on the reconstruction is barely visible. The features where the reconstruction issues are most prominent are plotted here. The good reconstruction of the anomalous sequence implies that the model did not learn the changed time constant well.

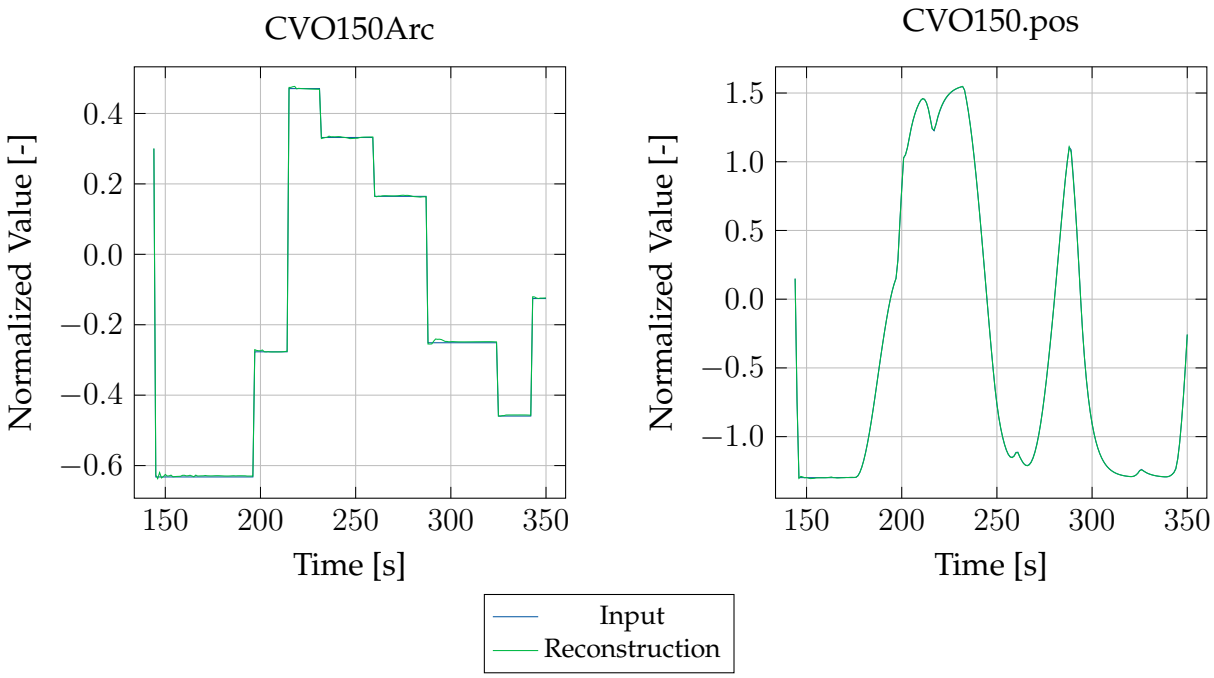


Figure 1: Reference tank pressure and position of the `CVO150` valve

LOX Leakage

While this fault takes place in the *Liquid Oxygen* (LOX) circuit, it influences the *Gaseous Nitrogen* (N2) circuit too as demonstrated by Figure 2. The influence is more prominent for the sensors in the LOX circuit though as plotted in Figure 3.

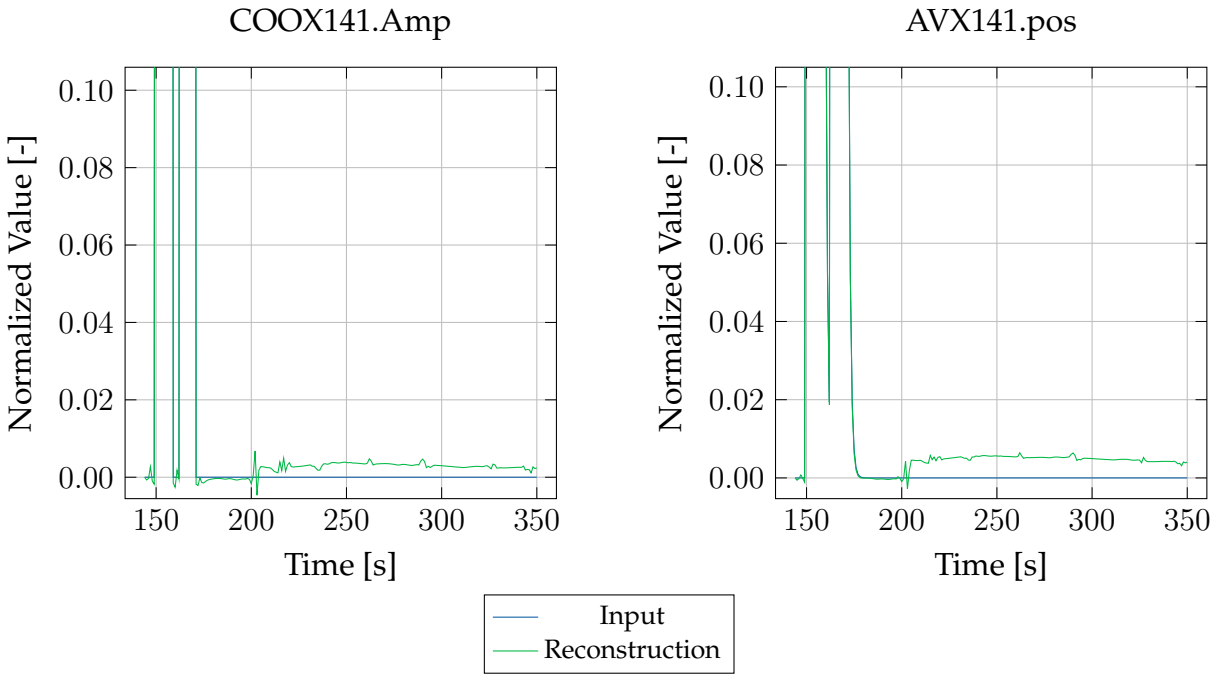


Figure 2: Command and position of the `AVX141` valve

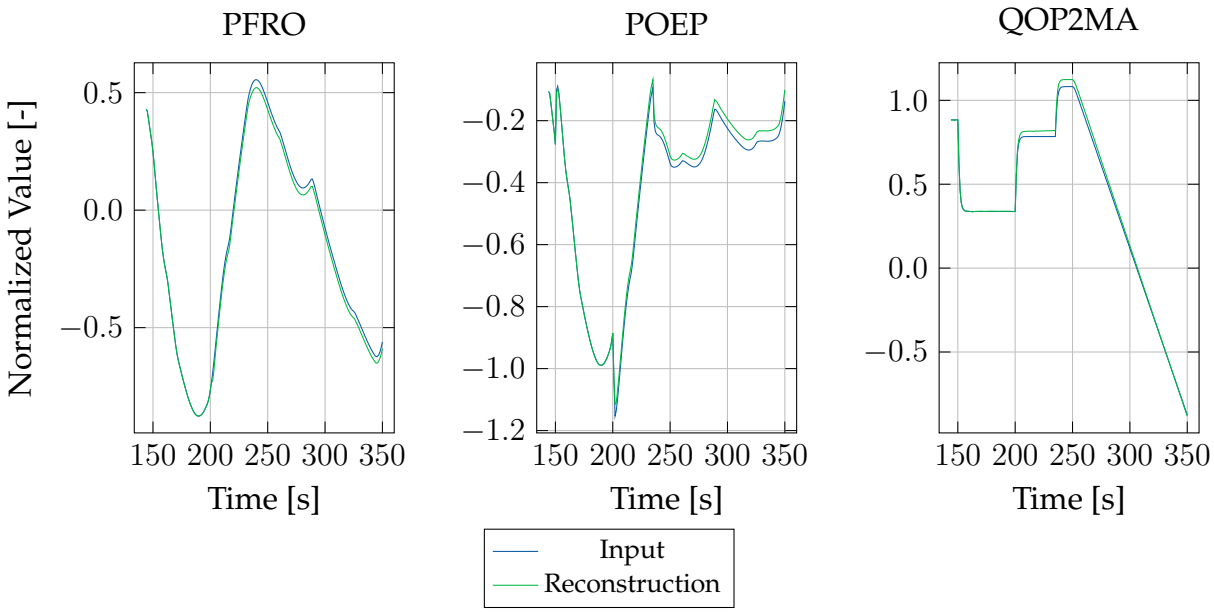


Figure 3: Pressures and mass flow in the LOX system

Sensor Drift

The AE "distributes" the error from one sensor to various features. It assumes some *negative* opening value of the `AVX141` valve which is very unrealistic but a clear indicator of an anomaly.

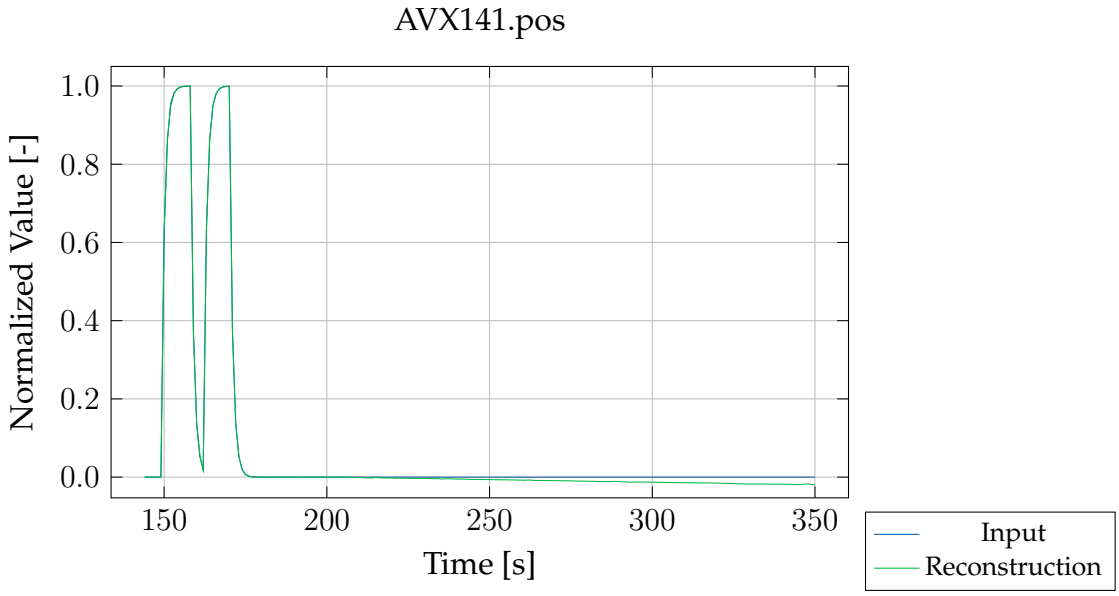


Figure 4: Command and position of the `AVX141` valve

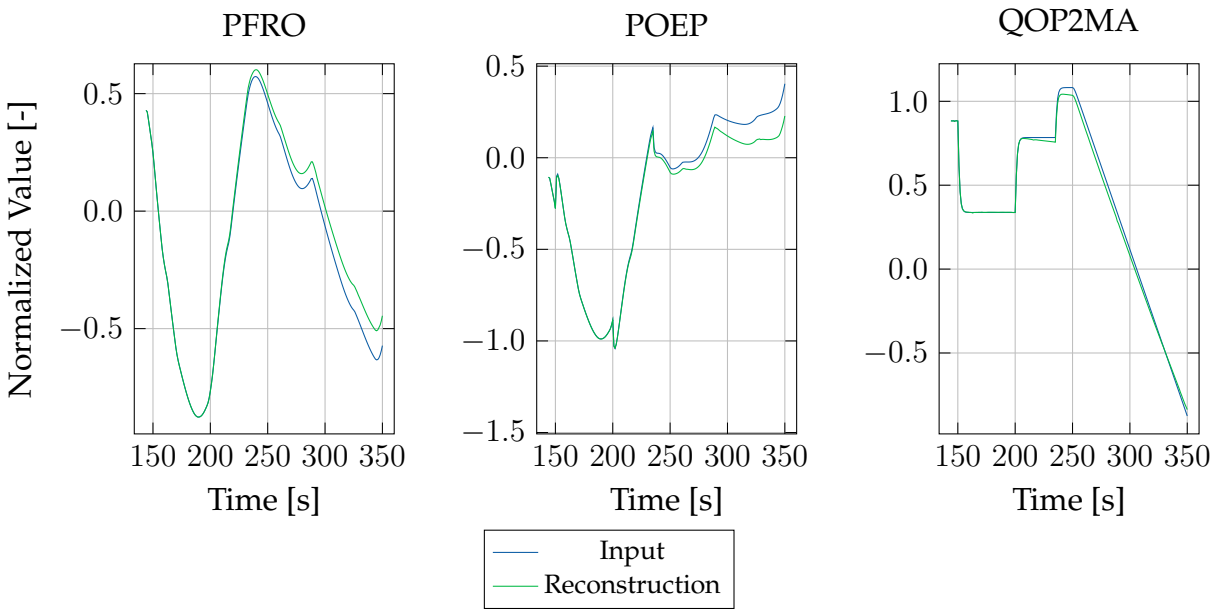


Figure 5: Pressures and mass flow in the LOX system

Sensor Frozen

A frozen `POEP` sensor influences the reconstruction of almost all features. The features with the strongest deviations are shown in Figure 6 and Figure 7.

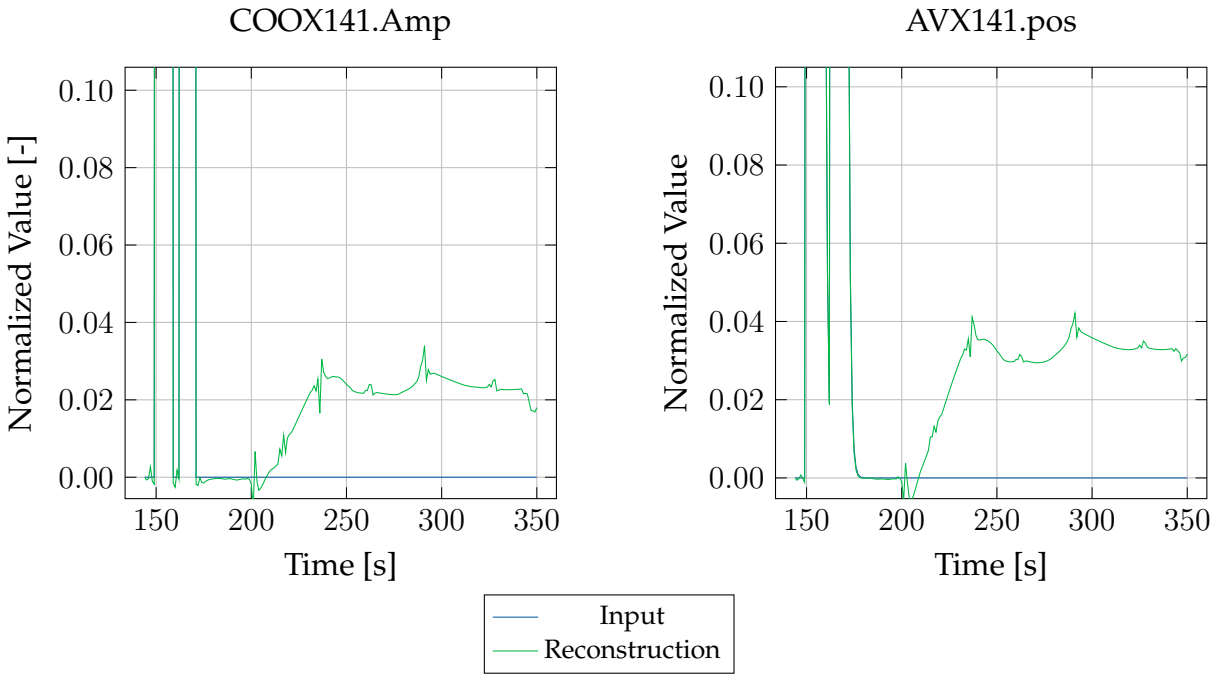


Figure 6: Command and position of the `AVX141` valve

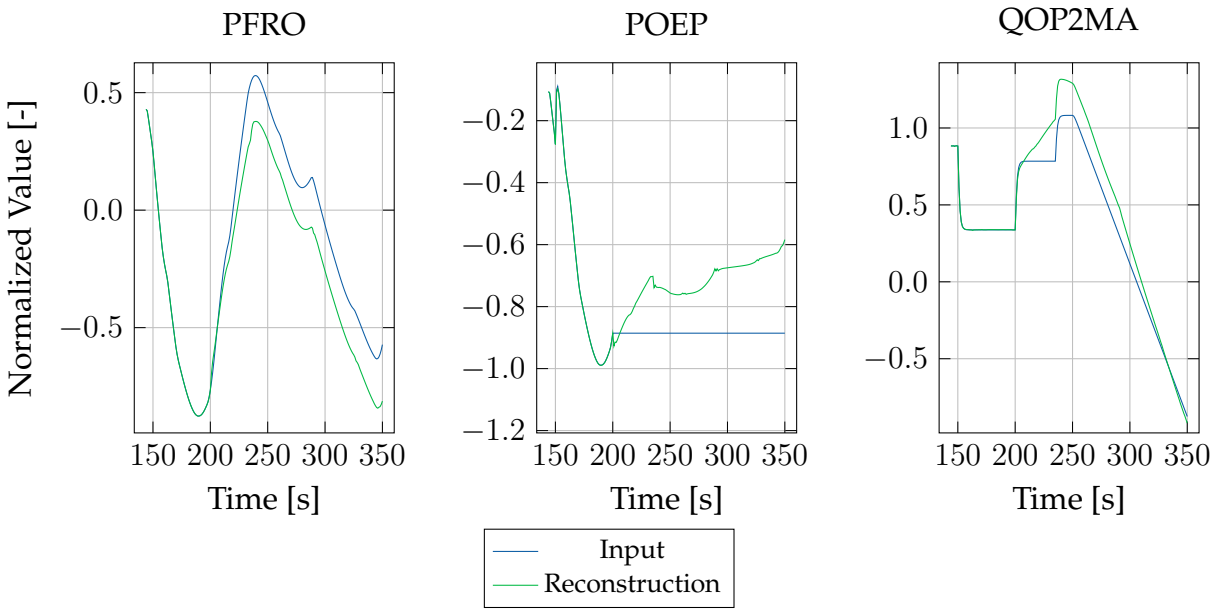


Figure 7: Pressures and mass flow in the LOX system

Sensor Offset

A sensor offset influences the system similar to a drifting sensor. However, the offset in the features stays almost the same over time.

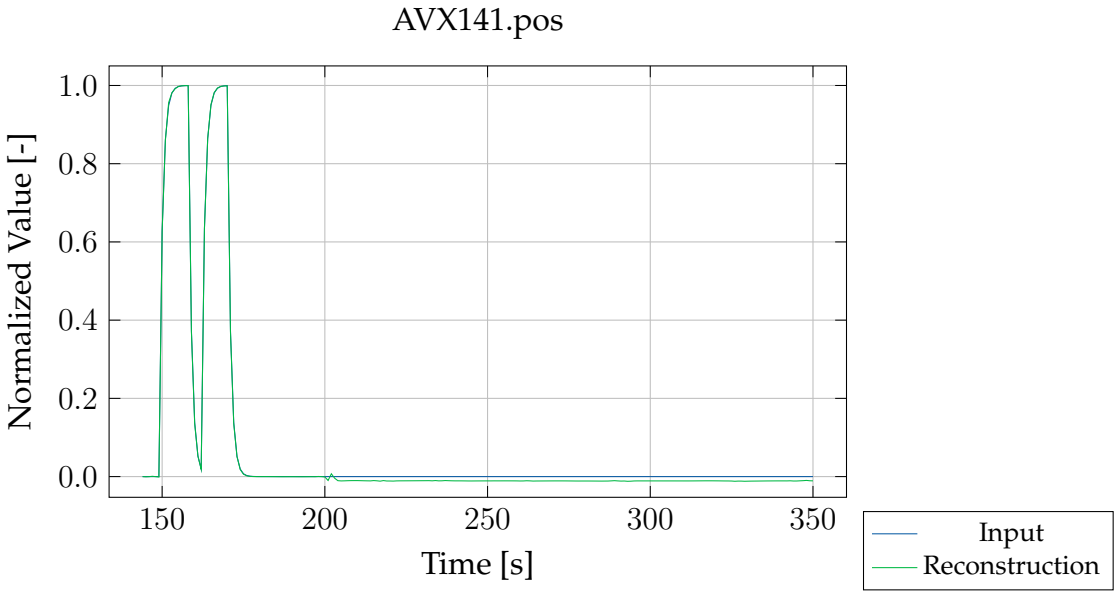


Figure 8: Command and position of the `AVX141` valve

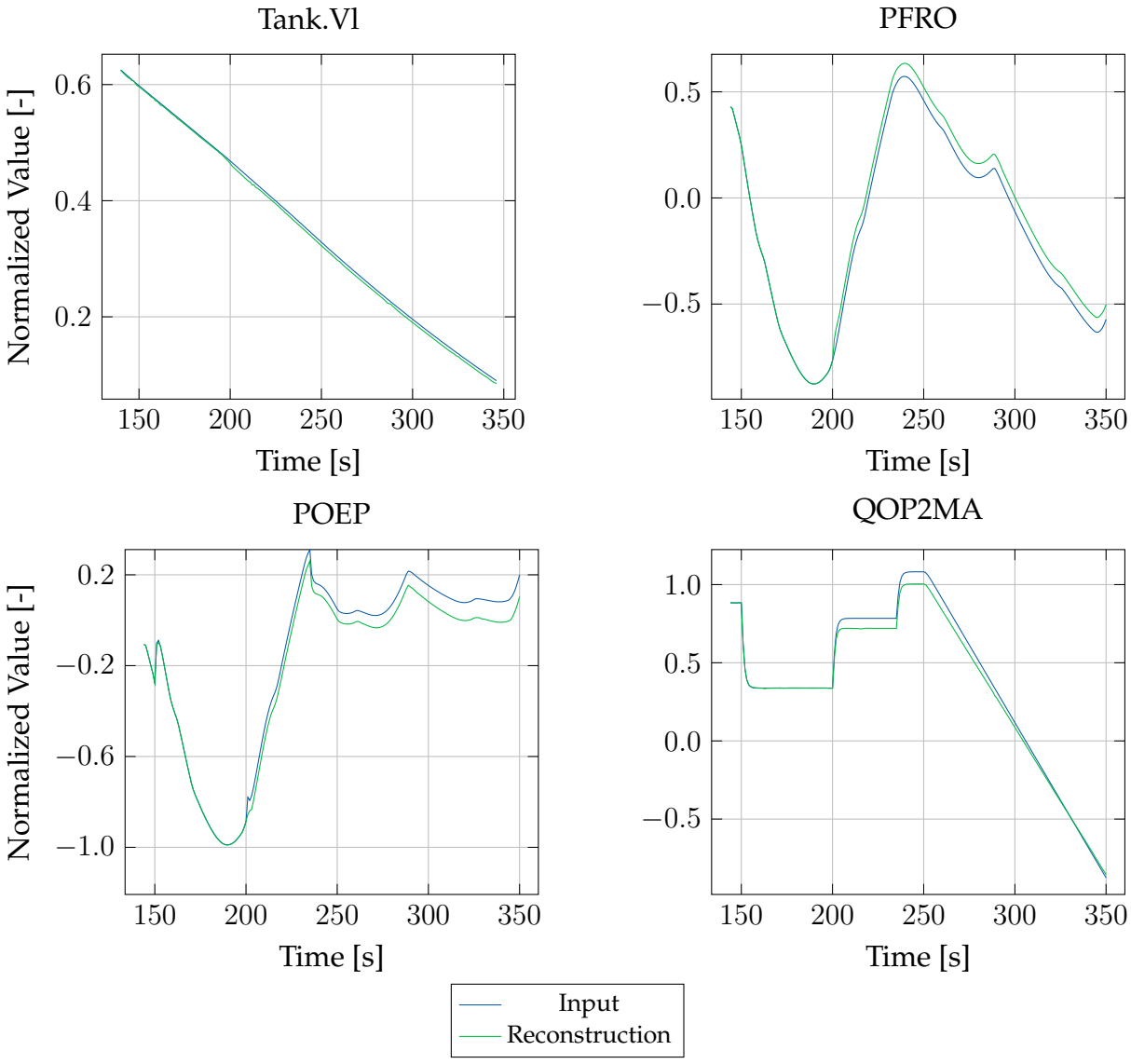


Figure 9: Volume of LOX in the tank, pressures, and mass flow in the LOX system

A.3 Confusion Matrices

The *confusion matrices* here are computed for the whole fault dataset or the runs corresponding to the fault types.

Table 1: Confusion matrices for *0 % False-Positive Rate on Validation Data*

(a) All faults

		Predicted	
		1	0
Actual	1	103922	4
	0	113078	139783

(b) Sensor frozen

		Predicted	
		1	0
Actual	1	28321	0
	0	15779	27363

(c) Sensor drift

		Predicted	
		1	0
Actual	1	32211	0
	0	11058	27857

(d) Sensor offset

		Predicted	
		1	0
Actual	1	43309	3
	0	52	28099

(e) LOX leakage

		Predicted	
		1	0
Actual	1	0	1
	0	42800	28197

(f) Changed τ_{CVO150}

		Predicted	
		1	0
Actual	1	81	0
	0	43389	27841

Table 2: Confusion matrices for *1 % False-Positive Rate on Faulty Data*

(a) All faults

		Predicted	
		1	0
Actual	1	171351	1397
	0	45649	138390

(b) Sensor freeze

		Predicted	
		1	0
Actual	1	40396	176
	0	3704	27187

(c) Sensor drift

		Predicted	
		1	0
Actual	1	41580	263
	0	1689	27594

(d) Sensor offset

		Predicted	
		1	0
Actual	1	43361	269
	0	0	27833

(e) LOX leakage

		Predicted	
		1	0
Actual	1	42555	363
	0	245	27835

(f) Changed τ_{CVO150}

		Predicted	
		1	0
Actual	1	3459	325
	0	40011	27516

Table 3: Confusion matrices for *Remove Spikes*

(a) All faults

		Predicted	
		1	0
Actual	1	169886	206
	0	47114	139581

(b) Sensor frozen

		Predicted	
		1	0
Actual	1	40368	21
	0	3732	27342

(c) Sensor drift

		Predicted	
		1	0
Actual	1	41601	20
	0	1668	27837

(d) Sensor offset

		Predicted	
		1	0
Actual	1	43361	42
	0	0	28060

(e) LOX leakage

		Predicted	
		1	0
Actual	1	42555	67
	0	245	28131

(f) Changed τ_{CVO150}

		Predicted	
		1	0
Actual	1	2001	56
	0	41469	27785

Table 4: Confusion matrices for *Flag Faulty*

(a) All faults

		Predicted	
		1	0
Actual	1	178850	1373
	0	38150	138414

(b) Sensor frozen

		Predicted	
		1	0
Actual	1	41278	303
	0	2822	27060

(c) Sensor drift

		Predicted	
		1	0
Actual	1	41009	0
	0	2260	27857

(d) Sensor offset

		Predicted	
		1	0
Actual	1	43361	459
	0	0	27643

(e) LOX leakage

		Predicted	
		1	0
Actual	1	42508	398
	0	292	27800

(f) Changed τ_{CVO150}

		Predicted	
		1	0
Actual	1	10694	213
	0	32776	27628

Table 5: Confusion matrices for *Parametric Thresholding*

(a) All faults

		Predicted	
		1	0
Actual	1	182242	40916
	0	34758	98871

(b) Sensor frozen

		Predicted	
		1	0
Actual	1	36671	7732
	0	7429	19631

(c) Sensor drift

		Predicted	
		1	0
Actual	1	42571	8322
	0	698	19535

(d) Sensor offset

		Predicted	
		1	0
Actual	1	43361	8297
	0	0	19805

(e) LOX leakage

		Predicted	
		1	0
Actual	1	42628	8089
	0	172	20109

(f) Changed τ_{CVO150}

		Predicted	
		1	0
Actual	1	17011	8150
	0	26459	19691

Table 6: Confusion matrices of *Peaks-Over-Threshold* (POT)

(a) All faults

		Predicted	
		1	0
Actual	1	162866	54134
	0	151	139636

(b) Sensor frozen

		Predicted	
		1	0
Actual	1	37458	6642
	0	21	27342

(c) Sensor drift

		Predicted	
		1	0
Actual	1	39999	3270
	0	28	27829

(d) Sensor offset

		Predicted	
		1	0
Actual	1	43361	0
	0	31	28071

(e) LOX leakage

		Predicted	
		1	0
Actual	1	40995	1805
	0	42	28156

(f) Changed τ_{CVO150}

		Predicted	
		1	0
Actual	1	1053	42417
	0	29	27812