Universiteit Leiden

Master Computer Science

[Feasibility Study for automatic regression analysis of Lithography Machine]

Name: [Manasvi Kumar]
Student ID: [S3938425]

Date: [04/03/2025]

Specialisation: [Data Science]

1st supervisor: [Dr. Hao Wang]
2nd supervisor: [Dr. Marcello Bonsangue]

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Contents

Acknowledgment

I sincerely thank my college supervisors, **Dr. Hao Wang** and **Dr. Marcello Bonsangue**, for their invaluable guidance, support, and encouragement during my internship and throughout the thesis project. Their expertise and insights have been instrumental in shaping the direction and outcomes of this work.

I am also profoundly grateful to the entire **Productivity Team at ASML** for allowing me to tackle a real-world data science problem. The team's talent, motivation, and collaborative spirit created an inspiring environment that greatly helped my learning experience. I owe special thanks to my mentor, **Daniel Machado**, Lithography System Performance Engineer at ASML, for supervising me throughout the internship, introducing me to the challenges related to throughput and the analytical processes undertaken by the team, and constantly guiding me with patience and expertise.

Additionally, I would like to thank **Egemen Guclu** for his support in helping me overcome roadblocks during the coding process, and **Jimi Hendriks** for engaging in insightful conversations about our work and its broader impact at ASML. My heartfelt thanks also go to **JongKoon Lim** and **Ghassan Charestan** for always looking out for me and encouraging me to push my boundaries and strive for excellence and also for organizing all the Team building activities where I got to make memories that I will cherish for a lifetime.

Finally, I extend my gratitude to my **Study Advisors at LIACS** for their guidance and support in helping me navigate and overcome the obstacles I encountered during the journey of my Master's Thesis.

This thesis would not have been possible without the contributions and support of all these individuals, and I am deeply thankful for their contributions and support throughout this journey.

Abstract

Background ASML[ASM25] is the largest supplier in the semiconductor industry as of 2023 and the sole provider globally of extreme ultraviolet lithography (EUVL) photolithography machines, which are essential for manufacturing the most advanced chips. However, with every new machine configuration, performance analysis becomes increasingly complex and requires significant engineers' effort. These lithography machines operate at the nanometer scale, where even the slightest variations in performance—potentially arising from normal operational factors—require detailed analysis to determine whether they signify genuine performance issues or anomalies.

Aim This study focuses on developing a utility to distinguish between genuine performance degradation and anomalies caused by outliers. The utility aims to streamline the analysis of lithography machine performance by comparing its operation in two states: a pre-state, representing its baseline configuration, and a post-state, reflecting its performance following software updates.

Method The utility integrates PyNomaly for probabilistic outlier detection and the Isolation Forest machine learning classifier for robust classification of anomalous data points. These methods collectively identify deviations that may impact performance trends. Additionally, the report discusses the implementation of test cases using Python's `unittest` package to ensure the reliability and robustness of the tool.

Results The results demonstrate that an optimal model combining PyNomaly and Isolation Forest effectively detects outliers at the first-level processes. Furthermore, a median deviation-based approach is proposed for analyzing anomalies in second-level processes. The methodology also incorporates test cases to validate the accuracy and reliability of the unsupervised anomaly detection models.

Conclusion This approach not only identifies anomalous data but also provides detailed insights into the performance of lithography machines at the second-level processes. The utility thus aids in reducing the time and effort engineers spend on performance evaluation, enabling a more efficient analysis workflow.

1 Introduction

In the modern era of advanced manufacturing and process automation, understanding and identifying anomalies in operational data has become a critical challenge.The semiconductor industry, particularly in the context of ASML's EUV lithography machines, encounters various anomalies that impact performance, yield, and reliability. These anomalies can arise due to issues such as Power fluctuations, Outlier durations in process steps etc. These anomalies can be caused due to environmental factors such as pressure, machine temperature, particle deposition on wafers etc.

ASML, a leading player in this domain, is at the forefront of developing advanced photolithography machines capable of operating at the nanometer scale. In this modern era of advanced manufacturing and process automation, understanding and identifying anomalies in operational data has become a critical challenge. This study focuses on ASML's EUV lithography machines, which are crucial for advanced semiconductor fabrication. Specifically, the work examines anomalous durations in machine processes that can indicate potential failures, inefficiencies, or deviations in the system. The problems at hand is that with growing number of machine configurations, analysis will become increasingly challenging, and thus by developing an anomaly detection utility tailored to ASML's lithography processes, this study aims to improve fault identification, reduce machine downtime, and enhance overall production efficiency. Anomalies, or outliers, represent data points that significantly deviate from the norm, often signaling potential issues or rare events that require further investigation. Analyzing such outliers is vital in industries that rely on precision and consistency, such as semiconductor manufacturing, where minor deviations in process durations can lead to significant disruptions or inefficiencies. The deviations in performance could be caused due to factors such as software updates, environmental conditions, and hardware wear. Identifying whether deviations in machine performance stem from genuine operational issues or anomalous data points is critical to ensuring consistent manufacturing quality.

1.1 Problem Statement

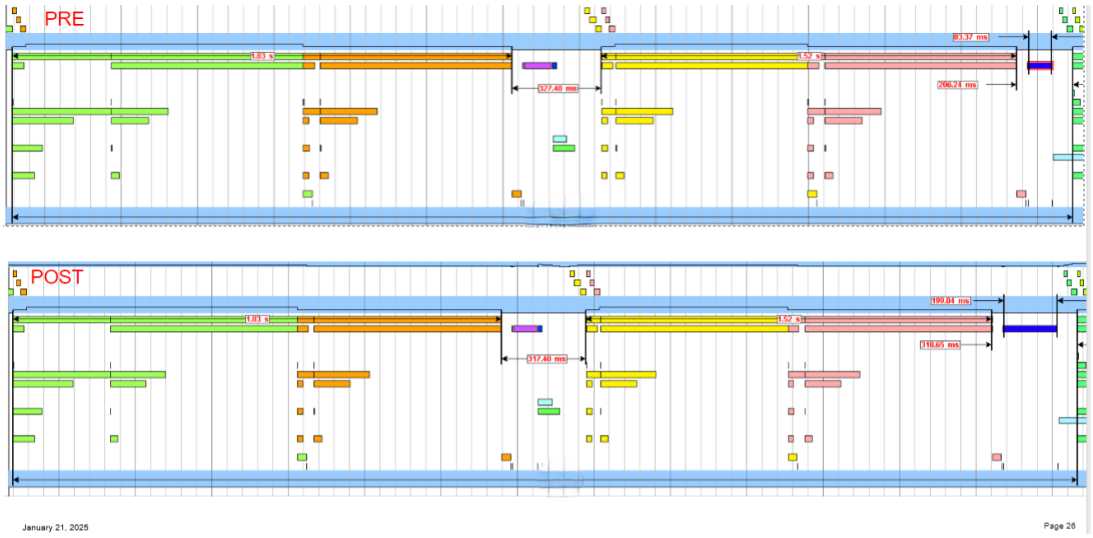

Figure 1: Processes in chip manufacturing inside Lithography machine

In complex industrial environments, identifying and managing outliers in process duration data remain a persistent challenge. Traditional methods often fail to adapt to the evolving patterns in data, leading to excessive false positives or overlooked anomalies. Performance evaluation becomes even more challenging when dealing with datasets from low-level processes, which often contain many data points, making manual analysis impractical. Engineers require automated tools to effectively distinguish outliers from meaningful performance variations, minimizing manual effort while maintaining accuracy. Moreover, the hierarchical nature of lithography processes introduces additional complexity, as data must be analyzed at both high and low levels, with nuanced relationships between these levels needing to be understood.

This study focuses on duration data, particularly non-stationary duration data, where the duration values change with time. These changes are typically affected due to factors such as software patches and environmental factors. Duration data is often continuous and unbounded, with extreme values being natural in some contexts but anomalous in others. Given the nature of this data, probabilistic methods that assign an anomaly score to each data point are particularly suitable.

The study addresses these challenges by developing a robust anomaly detection utility. The tool focuses on non-stationary duration data from lithography machines and leverages unsupervised machine learning models to identify outliers. The proposed approach combines probabilistic anomaly detection techniques with scalable machine learning algorithms for efficient and interpretable performance analysis.

1.2 Scope of the Study

This study focuses on evaluating lithography machine performance through a combination of probabilistic anomaly detection and machine learning classifiers. The dataset includes process

durations recorded in pre- and post-run states, divided into hierarchical levels to facilitate granular analysis. Key contributions of this study include:

A utility combining PyNomaly and Isolation Forest for robust outlier detection. Pre-check and preprocessing pipelines to ensure data consistency and integrity. Testing methodologies validate the proposed solution's performance and reliability, including input space partitioning and random testing. This work ultimately aims to streamline the workflow of ASML engineers by providing a reliable tool for identifying and analyzing anomalous performance trends in lithography machines.

2 Literature Review

Automatic anomaly detection has become critical today, where the sheer volume of data generated across industries makes manual tagging of outliers infeasible. Traditional supervised classification models require labeled datasets often unavailable in real-world scenarios. Moreover, these models rely on prior assumptions about the data, such as the distribution of classes, which may not hold for many complex datasets. In contrast, unsupervised learning methods are well-suited for anomaly detection as they do not require labeled data. These methods identify patterns, structures, and deviations directly from the data. By relying on intrinsic data properties, such as density, distance, or clustering tendencies, unsupervised methods isolate statistical outliers and predict that they are anomalies. The adaptability and scalability of unsupervised methods make them particularly useful in dynamic environments where data evolves.

Several unsupervised methods have been developed for anomaly detection. Clustering-based methods, such as K-means [LZS+10] and DBSCAN, are widely used in weather pattern tracking and fraud detection because they can group similar data points and flag instances that fall outside these clusters as anomalies. Clustering methods, although intuitive, may struggle with datasets containing overlapping clusters or varying densities. Statistical methods like the Z-test or Grubbs' test effectively detect univariate anomalies by identifying points that deviate significantly from the mean. However, they are less suitable for multivariate or high-dimensional data. Similarly, proximity-based techniques like k-nearest neighbors (k-NN) [TAP14] anomaly detection calculates the density around each point to flag low-density regions as potential outliers. These methods are computationally expensive for large datasets. Popular machine learning algorithms are One Class SVM [LHTX03], which uses a max-margin approach to separate normal data from anomalies, and Isolation Forest [LTZ08], which isolates anomalies through random partitioning. Autoencoders [ZP17], which reduce data dimensionality, are also effective, as they struggle to reconstruct outliers, making anomalies easier to identify. These algorithms are widely used in industrial anomaly detection, such as for predictive maintenance in manufacturing, While powerful and internet protocol network intrusion detection [SFAA23] are some of the applications. These methods often require careful parameter tuning and assumptions about data distribution to achieve optimal results.

Several previous works have also explored various approaches to anomaly detection in unlabeled duration data. For example, [BGCML21] provides a comprehensive overview of anomaly detection methods for duration data, emphasizing the importance of extracting outlier information

as useful data rather than simply cleaning the dataset. Another work [BM07] focuses on the problem of detecting unusual values or outliers from time series data where the underlying process is challenging to model. The authors propose two method variations that use the median from a neighborhood of a data point and a threshold value to compare the difference between the median and the observed data value. Both methods are computationally efficient and suitable for data streams such as sensor data on a machine.

In this thesis, we have chosen PyNomaly (LoOP) and Isolation Forest due to their ability to handle non-stationary duration data. PyNomaly provides probabilistic anomaly scores, making it well-suited for interpretability, while Isolation Forest efficiently isolates anomalies in large datasets without requiring explicit density estimation. We aim to improve the detection of anomalies in non-stationary duration data, ensuring both accuracy and interpretability. Our method implementation is validated through testing to provide a robust tool for ASML engineers when checking lithography machine performance data.

In the table below 1 we describe the different methods used in our study.

Method	**Strengths**	**Weaknesses**
One-Class SVM	Good for high-dimensional data	Computationally expensive
Isolation Forest	Scalable, efficient	Struggles with overlapping clusters
LOF	Adapts to density variations	Sensitive to parameter tuning
PyNomaly	Provides probabilistic anomaly scores	Assumes independence of data points

Table 1: Methods used in our study

3 Dataset

In this section, we discuss the sources of the datasets used in our study and the modifications performed to prepare the data for input into the models. We provide a detailed overview of the processing steps undertaken to transform the raw data into a usable format and the features obtained after cleaning. These steps are fundamental to our study, ensuring that the dataset is highly quality and the relevant features are optimized for machine learning modeling. Class imbalance was not a major concern in this dataset, as the proportion of anomalies was sufficient for effective model training and evaluation without requiring resampling techniques.

3.1 Data Source

The data used in this study was provided by the Productivity Department at ASML, one of the world's leading manufacturers of chip-making equipment. ASML designs and manufactures lithography machines, which are essential components in the chip manufacturing process.

The dataset comprises pre- and post-run durations of various processes involved in printing patterns on wafers. The data is further divided into two levels. A high-level process represents a major operational step, while low-level processes correspond to the sub-steps that contribute to it. For example, in a lithography machine, a high-level process might be 'wafer exposure,' while the low-level processes include 'alignment check,' 'light exposure,' and 'cooling phase.' We can consider a high-level process to be the engine of a car, and the lower-level processes are the individual parts inside the engine that together make up the engine.

The data for both levels includes a wide range of attributes, such as information on different batches and wafers for which these processes are performed. Additionally, the data records durations for both software and hardware processes. The diversity and size of the dataset make it well-suited for studying anomalies, as identifying outliers can provide valuable insights while analyzing the performance of lithography machines and help improve their operational efficiency.

3.2 Data Features

The dataset is structured into two levels, each with distinct features:

3.2.1 High-Level Processes Data

The data has the following attributes.

- **ID:** A unique identifier for each high-level process.
- **Name:** A descriptive name for the process, designed to be reader-friendly.
- **Batch:** The name of the batch for which the wafers are being processed.
- **Wafer :** The specific wafer that is being processed within the batch.
- **Duration:** The time taken to complete the high-level process.

3.2.2 Low-Level Processes Data

The data has the following attributes.

- **ID:** A unique identifier for each low-level process.
- **Batch:** The name of the batch for which the wafers are being processed.
- **Wafer:** The specific wafer being processed within the batch.
- **Process_Name1:** The initial segment of the process name.
- **Process_Name2:** The middle segment of the process name.

- **Process_Name3:** The final segment of the process name.

The segmentation of process names into three parts (Process_Name1, Process_Name2, Process_Name3) reflects the hierarchical or compositional nature of the low-level processes. It helps organize and analyze the data more effectively.

3.3 Data Processing

The machine data undergoes several preprocessing steps to convert the raw machine data into a format that is suitable for modeling. These steps ensure data consistency, eliminate errors, and enhance model performance. The process consists of two main stages: pre-check and processing.

The process consists of two main stages: pre-check and processing.

3.3.1 Pre-Check

The pre-check phase ensures that the raw data is consistent and complete before proceeding to further processing. The steps, shown here in Figure reffig:Precheck, include:

- **Shape of the Dataframe:** Ensuring that the pre-run and post-run data frames have matching attributes is crucial for consistency. If discrepancies exist, they may indicate data corruption or incorrect logging, requiring manual correction.
- **Number of Unique Processes:** Ensuring that the processes in the pre-run data match those in the post-run data.
- **Value of Unique Processes:** Comparing the string values of process names to confirm consistency between pre- and post-run data and to identify and eliminate duplicates.

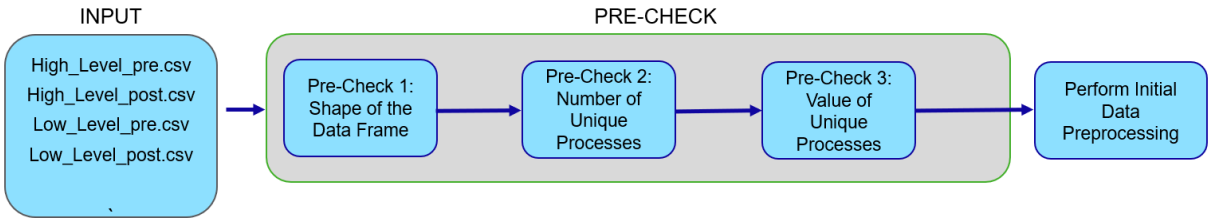

Figure 2: Data Pre-Check Pipeline

3.3.2 Processing

The processing phase involves cleaning and transforming the data to prepare it for machine learning. The pipeline is illustrated in Figure 3 and includes the following steps:

- **Mapping Data:** Converting data values to the appropriate data types required by the model.
- **Handling Nan Values**: Nan stands for Not A Number and is one of the common ways to represent the missing value in the data. It is a special floating-point value and cannot be converted to any other type than float. Nan value is one of the major problems in Data Analysis. It is very essential to deal with Nan values in order to get the desired results. Removing or imputing missing values to prevent errors and improve model accuracy. For this study, rows with Nan values were removed to maintain data integrity for the non-critical fields, and for the critical fields, we replaced the Nan value with the average for the remaining wafers.
- **Rounding off small values:** Values below a predefined threshold were rounded to zero to prevent overfitting. This threshold was determined based on the domain knowledge of ASML engineers.
- **Removing Numerical Prefixes**: Serial numbers attached to batch IDs were removed, as they do not contribute to the analysis and could introduce noise.

These preprocessing steps ensure that the dataset is clean, consistent, and optimized for anomaly detection in our machine-learning model.

Missing data and inconsistent formatting required significant preprocessing efforts. Additionally, the hierarchical nature of the data necessitated careful feature engineering to ensure compatibility with the machine learning model. Future work could explore alternative preprocessing techniques, such as automated feature selection or advanced imputation methods, to improve the use of the dataset.

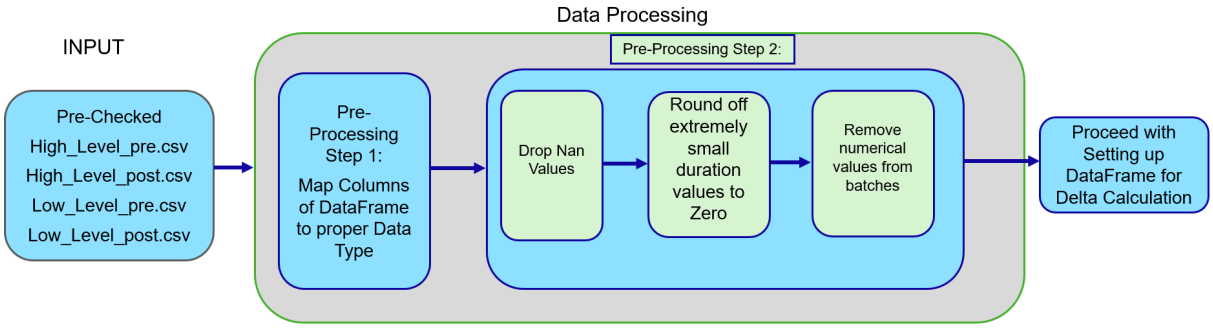

Figure 3: Data Processing Pipeline

4 Methods

In this section, we describe the various techniques explored for identifying outliers in the duration data. The methods combine statistical and machine learning approaches, each offering unique strategies for detecting anomalies.

We begin with histogram-based methods, which provide a visual and statistical approach to identifying deviations based on data distribution. The Inter Quartile Range (IQR) method is another statistical technique that detects outliers by examining the spread of the data.

Among the machine learning methods, the Local Outlier Factor (LOF) identifies anomalies by comparing the local density of data points. A One-Class Support Vector Machine (SVM) employs a boundary-based approach to classifying outliers in a feature space. The Isolation Forest Classifier uses a tree-based ensemble method to isolate and detect anomalous points effectively.

Finally, PyNomaly applies a probabilistic approach to anomaly detection, offering insights into how likely each data point is to be an outlier. Each method was evaluated for its effectiveness in detecting outliers in the lithography machine performance data.

In the following subsections, we provide a detailed explanation of each method, including their features and how they were applied in this study.

4.1 Histogram Method

Histogram-Based Outlier Scoring (HBOS) is a statistical anomaly detection method that evaluates outliers by constructing univariate histograms for each feature of the dataset. Unlike many algorithms, HBOS assumes feature independence, which makes it computationally efficient and scalable for large datasets with high-dimensional features. This efficiency is particularly advantageous in scenarios where a large volume of data needs to be analyzed, such as network security or industrial systems.[GD12] This data is usually high dimensional where feature independence can be assumed, making it computationally efficient for large-scale anomaly detection.

For numerical features, HBOS employs two techniques for constructing histograms: static bin width and dynamic bin width. Static binning divides the range of values into equal intervals, while dynamic binning adjusts the bin width based on the data distribution. Dynamic binning is particularly effective when the dataset contains gaps or exhibits long-tail distributions, as it adapts to the data's inherent structure. The number of bins, k, is typically set to the square root of the number of instances, N, to balance granularity and computational cost.

To compute the HBOS score, the algorithm normalizes the histograms so that the maximum bin height equals 1.0, ensuring equal weighting across features. The score, for instance, is calculated as the sum of the logarithmic inverses of the density estimates for each feature, as follows:[PB19]

$$HBOS(p) = \sum_{i=1}^{d} \log\left(\mathrm{hist}(p_i)\right)$$

where d is the number of features, p is a data point, and $\mathrm{hist}(p_i)$ is the normalized histogram height of of the ith feature.

HBOS offers several advantages, including its fast computation time, the absence of a training

phase, and the ability to assign a continuous outlier score to each instance. These properties make it a practical and effective choice for anomaly detection tasks, particularly in applications involving high data volume or time-sensitive analysis.

4.2 Interquartile Range Method

The Interquartile Range (IQR) method is a statistical technique used for detecting anomalies by measuring the variability of data [Wik24a]. It divides the dataset into four equal parts using quartiles, which are specific points that split the data when arranged in ascending order. The first quartile (Q1) represents the 25th percentile of the data, below which 25% of the observations lie, while the third quartile (Q3) represents the 75th percentile, below which 75% of the observations lie. The interquartile range is defined as the difference between these two quartiles and measures the spread of the middle 50% of the data:

$$\text{IQR} = Q3 - Q1$$

Using the IQR, outliers are identified as data points that fall outside a specified range defined by the lower and upper bounds. These bounds are calculated as follows[WWLT14]:

$$\text{Lower Bound} = Q1 - k \cdot \text{IQR}$$
$$\text{Upper Bound} = Q3 + k \cdot \text{IQR}$$

Here, k is a constant that determines the sensitivity of the detection. A typical value for k is 1.5, although it can be adjusted based on the requirements of the analysis or the characteristics of the dataset. Any data point lying below the lower bound or above the upper bound is classified as an outlier.

The IQR method is widely appreciated for its robustness to extreme values. It does not depend on the mean or standard deviation, both of which can be influenced by outliers. Additionally, it is simple to implement and interpret, making it an effective tool for exploratory data analysis.

However, the method has some limitations. It assumes that the data is symmetrically distributed and may not perform well with skewed or multimodal datasets. In such cases, the computed bounds might fail to capture all anomalies or misclassify normal points as outliers.

In summary, the IQR method provides a straightforward and robust approach to anomaly detection by leveraging the distribution of data. Its simplicity and resistance to the influence of extreme values make it an essential technique for identifying potential outliers in various datasets. IQR assumes a symmetric distribution, which may lead to misclassification of anomalies in skewed datasets.

4.3 Local Outlier Factor

The Local Outlier Factor (LOF) is an anomaly detection technique that identifies outliers based on their local density relative to their neighbors. Unlike global methods, LOF focuses on the local neighborhood of data points, making it effective for datasets where anomalies are more evident in certain regions than others. The LOF depends on the a priori given number of neighbors, making it difficult to balance sensitivity and specificity in anomaly detection.

The LOF algorithm assigns an outlier score to each data point by comparing its local density with the densities of its neighbors. A data point with a significantly lower local density than its neighbors is considered an outlier. The algorithm involves the following steps[AASM20]:

1. Compute the k-distance: For a given point p, the k-distance is the distance to its k-th nearest neighbor. This parameter k determines the size of the local neighborhood and must be chosen carefully based on the dataset's characteristics.

2. Define the neighborhood: The k-distance neighborhood of p includes all points within the k-distance of p.

3. Calculate the local reachability distance: The local reachability distance of p with respect to a neighbor q is defined as:

$$\text{reachability\_dist}_k(p, q) = \max\{\text{k-distance}(q), \text{distance}(p, q)\}$$

This metric prevents the distance to very close neighbors from being too small, which could distort the density computation.

4. Determine the local density: The local density of p is the inverse of the average reachability distance of p to all its neighbors:

$$\text{lrd}_k(p) = \frac{1}{\frac{\sum_{q \in N_k(p)} \text{reachability\_dist}_k(p,q)}{|N_k(p)|}}$$

Here, $N_k(p)$ represents the set of k-distance neighbors of p [Wik24c].

5. Compute the Local Outlier Factor: Finally, the LOF score for p is calculated as the ratio of the average local density of its neighbors to its own local density:

$$\text{LOF}_k(p) = \frac{\sum_{q \in N_k(p)} \text{lrd}_k(q)}{|N_k(p)| \cdot \text{lrd}_k(p)}$$

A higher LOF score indicates that p is an outlier relative to its neighbors [Wik24c].

The LOF method has several advantages. It is highly adaptable to data with varying density regions and does not assume any specific data distribution. It is particularly effective in cases where the global density approach might fail, such as datasets with clusters of different densities[BKNS00].

However, LOF also has limitations. The method's performance depends on the choice of k, which must balance capturing local structures without introducing noise. Additionally,

LOF can be computationally expensive for large datasets, as it requires calculating pairwise distances[APB18].

In conclusion, the Local Outlier Factor is a robust and flexible anomaly detection technique that identifies outliers by comparing local densities. Its ability to adapt to local data characteristics makes it a powerful tool for analyzing complex datasets with non-uniform distributions.

4.4 One-Class SVM

The One-Class Support Vector Machine (One-Class SVM) is a machine learning algorithm used for anomaly detection. It identifies outliers by learning the boundary that separates normal data points from anomalies. This method is particularly effective in scenarios where the dataset consists mostly of normal data with only a few anomalous instances.

One-class SVM is built on the principles of Support Vector Machines (SVMs), which are primarily used for classification and regression tasks. In the case of One-Class SVM, the algorithm learns a decision function $f(x)$ that distinguishes the regions in the feature space where most of the data points lie (normal instances) from the regions where few or no points exist (outliers)[HS13].

The One-Class SVM algorithm works as follows:

1. Kernel function: A kernel function $K(x, y)$ computes the inner product between the image of two data points x, y under a (nonlinear) feature map, which makes it easier to separate the normal instances from outliers.

Commonly used kernels include the Radial Basis Function (RBF) kernel [SWS+99]

$$K(x, y) = \exp\left(-\gamma \|x - y\|^2\right)$$

where γ is a hyperparameter that controls the smoothness of the decision boundary.

2. Optimization objective: The algorithm attempts to maximize the margin between the origin (representing anomalies) and a hyperplane that encloses most of the normal data points. The optimization problem can be formulated as [SEK05]:

$$\min_{w,\xi,\rho} \frac{1}{2}\|w\|^2 + \frac{1}{\nu N}\sum_{i=1}^{N} \xi_i - \rho$$

subject to:

$$w^T \phi(x_i) \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, N$$

Here:

- w represents the weights of the hyperplane,
- $\phi(x_i)$ is the feature map,
- ρ is the offset of the hyperplane,

- ξ_i are slack variables allowing for some misclassification and
- ν is a parameter that controls the trade-off between maximizing the margin and the fraction of anomalies allowed.

3. Decision function: Once the model is trained, the decision function is given by [SWS+99]:

$$f(x) = \sum_{i=1}^{N} \alpha_i K(x_i, x) - \rho$$

where α_i are the support vector coefficients. A point x is considered normal if $f(x) \geq 0$ and anomalous if $f(x) < 0$.

Key Features and Advantages: - Unsupervised Learning: One-Class SVM does not require labeled data for training, making it suitable for anomaly detection in real-world scenarios where labels are often unavailable. - Flexible Kernel Selection: The choice of kernel allows the method to capture complex patterns and adapt to various data distributions[CQS13]. - Scalability: Efficient optimization techniques enable One-Class SVM to handle moderately large datasets.

One-class SVM has several limitations. Its performance is highly sensitive to the choice of hyperparameters such asν and γ, which often require careful tuning to achieve optimal results. Additionally, the method can be computationally expensive for very large datasets due to the need for pairwise kernel computations. Furthermore, One-Class SVM assumes that anomalies are rare, making it less effective when the fraction of anomalies in the dataset is not small.

In summary, One-Class SVM is a powerful anomaly detection technique that uses SVM principles to separate normal instances from anomalies in the feature space. Its flexibility and adaptability make it a widely used method for unsupervised anomaly detection.

4.5 Isolation Forest

Isolation Forest is an anomaly detection algorithm that isolates anomalies instead of profiling normal data points. It is based on the idea that anomalies are "few and different" compared to normal instances, making them easier to isolate. This method is particularly efficient for large datasets due to its linear time complexity[LTZ08].

The Isolation Forest algorithm partitions the data by constructing a set of random decision trees called isolation trees. This isolation is achieved by recursively splitting data points along random feature values. Because anomalies are rare and distinct, they are more likely to be isolated earlier in the process, resulting in shorter path lengths in the tree structure. The Isolation Forest is selected due to its linear time complexity and robustness to high-dimensional data, making it more efficient than traditional decision trees for anomaly detection.

Algorithm Steps: 1. Random Subsampling: A subset of the data is randomly selected to construct each isolation tree. This subsampling reduces the computational cost and improves efficiency [LTZ12]. 2. Tree Construction: - For each tree, the dataset is recursively partitioned by randomly selecting a feature and a split value within the feature range. - The splitting

continues until all points are isolated (i.e., belong to their own partition) or the tree reaches a predefined height. 3. Anomaly Scoring: - The path length of a point x in an isolation tree is defined as the number of splits required to isolate the point. The anomaly score is computed based on the average path length across all trees. Shorter path lengths correspond to higher anomaly scores, as anomalies are easier to isolate.

The anomaly score for a point x is given by [Wik24b]:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$$

where: - $E(h(x))$ is the average path length of x across all trees, - $c(n)$ is the average path length of unsuccessful searches in a Binary Search Tree of n points, approximated as [LTZ08]:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n},$$

with $H(i)$ being the i-th harmonic number ($H(i) = \ln(i) + \gamma$, where γ is the Euler-Mascheroni constant).

A score close to 1 indicates a high likelihood of the point being an anomaly, while a score close to 0 suggests the point is normal.

Key Features and Advantages: - Unsupervised Learning: Isolation Forest does not require labeled data, making it suitable for a wide range of anomaly detection tasks. - Linear Time Complexity: The algorithm has a linear computational complexity, $O(n \cdot \log(n))$, where n is the number of data points. - Scalability: It can handle large datasets efficiently due to its random subsampling and simplicity. - Robustness: Isolation Forest performs well on datasets with high-dimensional and heterogeneous data.

Isolation Forest has certain limitations. Its performance can vary across runs due to the inherent randomness in tree construction unless a random seed is fixed to ensure consistent results. Additionally, the method requires careful tuning of hyperparameters, such as the number of trees and the subsample size, to achieve optimal performance. Furthermore, Isolation Forest assumes that anomalies are sparse and distinctly separated from the normal data, which may limit its effectiveness in datasets where this assumption does not hold.

In summary, Isolation Forest is a powerful and efficient algorithm for anomaly detection. By leveraging the concept of isolation rather than density or distance, it offers a fast and scalable solution for identifying anomalies in large and complex datasets. Its simplicity and robustness make it one of the most widely used anomaly detection techniques in practice.

4.6 Pynomaly

Pynomaly is a probabilistic anomaly detection method designed to identify outliers in numerical sequences. It leverages the statistical properties of data to calculate the likelihood of each point being an anomaly. This method is particularly useful when working with time series or sequential data, as it takes into account the underlying distribution and patterns within the data. PyNomaly provides probabilistic anomaly scores, making it useful for interpretability and ranking outliers based on confidence levels.

Methodology: Pynomaly is based on a statistical model that assumes a parametric or non-parametric distribution of the data. It calculates an anomaly probability score for each data point using likelihood-based techniques. The primary steps of the algorithm are as follows[Con18]:

1. Data modelling: - Pynomaly estimates the probability density function (PDF) of the data. This can be achieved using parametric methods such as Gaussian distribution fitting or non-parametric approaches like kernel density estimation (KDE). - The PDF describes the likelihood of observing a specific value in the dataset.

2. Anomaly Scoring: - For each data point x, the probability of observing x is calculated using the estimated PDF, denoted as $P(x)$. - To identify anomalies, Pynomaly assigns a score to each data point based on its probability. Points with very low probabilities are more likely to be anomalies. - The anomaly score is inversely proportional to the probability:

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

where higher scores indicate a greater likelihood of being an anomaly.

3. Thresholding: - A user-defined threshold is applied to the anomaly scores to classify points as anomalies or normal. The threshold determines the sensitivity of detection, with lower thresholds identifying more anomalies.

Key Features and Advantages: - Probabilistic Framework: By estimating the probability distribution, Pynomaly provides a robust way to quantify the likelihood of data points being anomalous. - Flexibility: The method supports various types of distributions, making it adaptable to different datasets and contexts. - Continuous Scoring: Unlike binary classification methods, Pynomaly provides a continuous anomaly score, allowing for a more nuanced interpretation of results.

PyNomaly has several limitations that should be considered. The method assumes that data points are drawn independently from the underlying distribution, an assumption that may not hold true for datasets with high correlations among variables. Additionally, its accuracy in detecting anomalies is highly sensitive to the choice of the probability distribution used to model the data, which requires careful consideration. Finally, for large datasets, estimating the probability density function (PDF) can be computationally expensive, especially when using non-parametric approaches such as kernel density estimation (KDE).

Pynomaly is well-suited for detecting anomalies in time series data, sensor readings, or any sequential datasets where probabilistic modeling can capture the inherent data characteristics. Its probabilistic nature allows it to be combined with other methods, such as Isolation Forest or clustering techniques, for hybrid anomaly detection approaches.

In conclusion, Pynomaly is a versatile and robust anomaly detection method that combines the strengths of statistical modeling and probabilistic scoring. Leveraging data distributions enables the effective detection of anomalies in various types of datasets while providing interpretable results through continuous scoring.

5 Experimental Design

The goal of our study is to design a system capable of detecting outliers in duration data. Hence, we need to evaluate the categorization ability of various classification models and understand the impact of outliers on duration data, as explored through machine learning and statistical methods. By identifying the outliers, we can provide quantitative evidence to get better insights into data and explore whether there are certain outliers that are common for some particular processes.

In this section, we explain how we designed experiments to evaluate the effectiveness of our anomaly detection methods in identifying outliers in lithography machine performance data. The objective is to compare different methods and setups based on their accuracy, precision, recall, and F1 score and determine the most suitable approach for real-world deployment.

In figure 4, we show the different processes that happen in the photolithography machine, and in the figure, a particular process highlighted with a red color border is marked for having a delta. However, on further analysis, it is concluded that this delta is caused by an anomaly and is not an issue due to the deployed software changes. However, this analysis was done manually by an engineer, and the goal of this study is to develop a utility that detects these anomalies automatically using the methods proposed earlier.

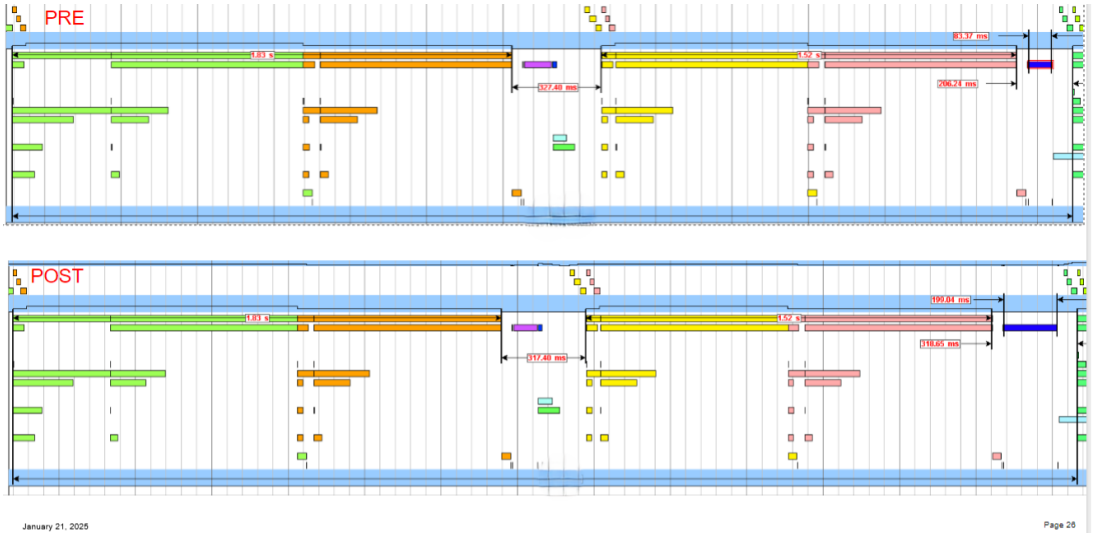

Figure 4: Gantt chart of processes. The bars represent the different processes that happen in Lithography machine while manufactuing a chip.

5.1 Experimental Setup

The system is implemented in Python 3.11 using algorithms from sklearn [PVG+11] an an open-source machine-learning library that provides a wide range of tools for data preprocessing and model evaluation and PyNomaly [Con], which performs anomaly detection using LoOP: Local Outlier Probabilities, a local density-based outlier detection method providing an outlier score in the range of [0,1]. The experiments were conducted on datasets provided by the ASML

productivity team. The data is obtained from running the photolithography machine for two scenarios, first without the software changes, which is the pre-run, and then after deploying the software changes, which is the post-run. As stated earlier, the data consists of two-level processes, first the high level and second the low level, and is preprocessed and cleaned before being fed into the models.

5.2 Evaluation Metrics

In Table 2, we can see the metrics used for the evaluation of the performance of all the methods

Metric	**Description**
Accuracy	The proportion of correctly classified instances among the total instances.
Precision	The ratio of correctly identified positive instances to the total predicted positive instances.
Recall	The ratio of correctly identified positive instances to the total actual positive instances.
F1 Score	The harmonic mean of precision and recall, balancing false positives and false negatives.

Table 2: Metrics used for Evaluation

5.3 Hyperparameter Tuning

Hyperparameter tuning is a critical step in optimizing machine learning models, as it directly influences the model's performance and generalizability. Selecting the right hyperparameters can significantly improve the ability of the model to detect anomalies effectively while reducing overfitting.

In this study, we employed the `GridSearchCV` method from the `sklearn` library to systematically explore a range of hyperparameter combinations and identify the most optimal configuration for each model. `GridSearchCV` performs an exhaustive search over specified parameter values by training and validating the model on a cross-validated subset of the data. This ensures that the selected hyperparameters generalize well to unseen data, improving model robustness.

The grid search process involves defining a parameter grid, where each hyperparameter is assigned a range of possible values. The method then evaluates all combinations in the grid, leveraging cross-validation to compute performance metrics for each configuration. Finally, it selects the combination that maximizes the chosen evaluation metric, ensuring the best trade-off between precision, recall, and overall accuracy. In the Table 3 we show the optimal values of the hyperparmaters obtained for our models and which we use to initialize our models with before the evaluation. The performance of the models varied significantly based

on different hyperparameter values. For instance, in Isolation Forest, increasing the number of trees improved anomaly detection at the cost of higher computational time, while adjusting the contamination parameter affected the sensitivity of outlier detection. In One-Class SVM, lower values of ν led to fewer anomalies being detected, whereas higher values increased false positives. Similarly, for Local Outlier Factor, tuning the neighborhood size (k) influenced how local density variations were captured—smaller values made the model more sensitive to minor fluctuations, while larger values resulted in smoother decision boundaries. These variations highlight the importance of careful hyperparameter tuning to balance detection accuracy and computational efficiency.Appendix 8provides an overview of the hyperparameters explored for the three different models under investigation. For some particular hyperparameters, such as n_neighbors, the value was set to 10, as this is the number of wafers that are processed by the photolithography machine during each run, and we obtain the duration data for each of these 10 wafers. The contamination hyperparameter is set to auto, which incorporates randomness in the model and helps avoid overfitting.

By systematically tuning hyperparameters using `GridSearchCV`, we ensured that our models achieved optimal performance across diverse datasets and anomaly detection scenarios.

5.3.1 Training Time

The computation time required by the models scales significantly with the increase in data. For training, according to the current process, we use the $fit\_predict$ methods from the models, and due to this, the training time is significantly less.

Methods	Best Hyperparameters
Local Outlier Factor	n_neighbors = 10 contamination = auto algorithm = [auto, ball_tree] leaf_size = 30
One-Class SVM	nu = 0.01 kernel = rbf gamma = auto
Isolation Forest	n_estimators = 100 max_samples = 10 contamination = auto
PyNomaly	extent = 1 n_neighbors = 10

Table 3: Best Hyperparameters for the Methods

5.4 Performance Metrics Comparison across Systems

To evaluate the efficiency and effectiveness of the outlier detection methods across different systems, a series of experiments were conducted using the optimal hyperparameters identified through GridSearchCV. Specifically, we tested the performance of the models outlined in previous sections, including Local Outlier Factor, One-Class SVM, Isolation Forest, and Pynomaly. For consistency and fairness, all models were evaluated using the same experimental setup and datasets. The comparison metrics, including Accuracy, Precision, Recall, and F1 Score, were carefully analyzed to assess the strengths and limitations of each system. The detailed results of this evaluation are presented in Table 4, while visualizations summarizing the comparative performance in the form of Confusion Matrices are provided in Figure8.

5.4.1 Results and Discussion

Precision and Recall are chosen as primary evaluation metrics because anomaly detection often involves imbalanced datasets, where accuracy alone can be misleading. From the table 4 we can see that while PyNomaly achieves the highest overall F1-score, Isolation Forest exhibits better recall, making it more suitable for applications where detecting all anomalies is critical. Furthermore, another aspect that we focus on is avoiding overfitting of our models. This is achieved by setting the $contamination$ parameter of the **Isolation Forest** model to $auto$; this ensures random splitting for each internal node of an iTree and thus reduces overfitting. Finally, we cannot provide any baseline comparison since. Currently, the analysis is done completely in a manual fashion, and the utility developed in our study is a first step towards the automation of the analysis process.

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
PyNomaly	**0.962**	**0.973**	**0.947**	**0.951**
One-Class SVM	0.785	0.794	0.761	0.73
Isolation Forest	0.90	0.92	**0.96**	0.94
Local Outlier Factor	0.87	0.93	0.912	0.924

Table 4: Performance Comparison of different models. The scores for the highest-performing model are highlighted in Bold.

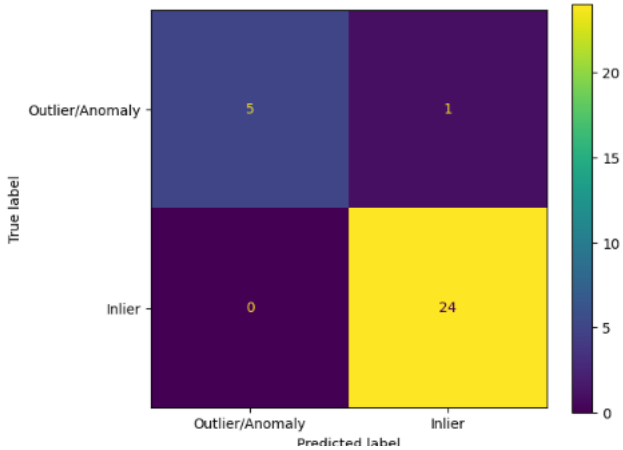

Figure 5: PyNomaly

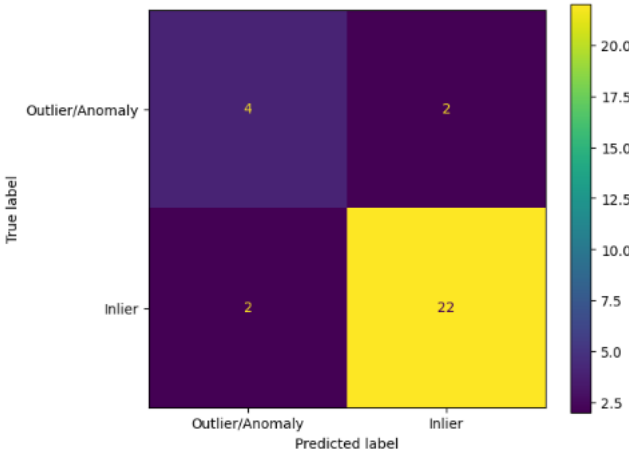

Figure 6: Local Outlier Factor

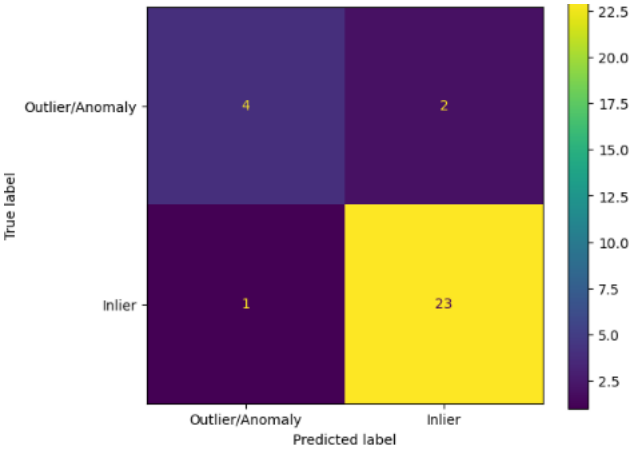

Figure 7: Isolation Forest

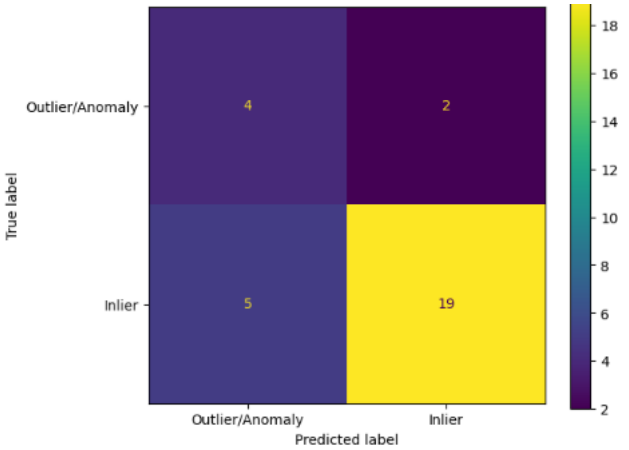

Figure 8: One-Class SVM

As is clear from the results, PyNomaly Outperformed the other three models in majority of the four metrics, hence we select PyNomaly as the final model.

6 Testing

Following the experimental evaluation of different anomaly detection methods, verifying the correctness of our framework ensures that our implementation performs reliably under diverse conditions. We use software testing techniques to validate the implementation of our anomaly detector framework. The primary focus is on assessing whether our code is correct, meaning that it correctly finds and distinguishes between genuine anomalies and is not influenced by normal variations in lithography machine performance. Software Testing is a critical aspect of any system to ensure its reliability, robustness, and accuracy. It enables the identification of flaws or unexpected behaviors in the implementation and helps validate the correctness of the methods employed. By systematically evaluating the system under various scenarios, testing builds confidence in the results and ensures that the system performs as intended across diverse conditions. This section discusses the testing methodologies used in this work, starting with input space partitioning and followed by random testing.

6.1 Input Space Partitioning

Input space partitioning was employed to ensure robust testing of the implemented methods because it allows for systematic evaluation of the code across different scenarios, ensuring that anomalies are detected consistently regardless of data distribution. The data was systematically divided into distinct partitions to evaluate performance under various scenarios. The first division involved separating the data into two subsets: one containing outliers and the other devoid of outliers. This division allowed for precise testing of the system's ability to handle both anomalous and normal data distributions[Pan99]. We also tested the code for the edge cases wherein the input was divided into extremely large and small durations and also on anomalies that were extremely close to normal variation, which might be harder to distinguish.

Additionally, further partitions were created based on specific lithography machines. For each machine, two datasets were prepared—one containing outliers and the other without. This machine-specific partitioning enabled tailored testing for each system's unique characteristics and potential variances in data behavior [AO17].

The testing process was implemented using Python's `unittest` package [sf], ensuring a structured and automated testing framework. This setup facilitated consistent validation of the methods against the defined partitions, ensuring the correctness and reliability of the implemented solutions.

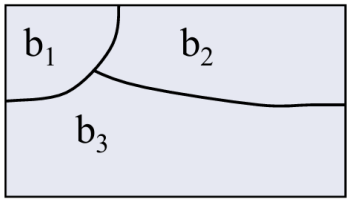

Figure 9: Input Space Partitioning

6.1.1 Results and Discussion

Expected results were generated based on domain experts' knowledge of previous system logs. These were then compared with the actual values computed by our implementation. The results from our implementation matched the expected results.

6.2 Random Testing

Random testing was conducted as an auxiliary approach to supplement the primary testing methodology. It serves as an additional mechanism to input space partitioning to validate certain aspects of the system[Ham94].

The implementation of random testing focused on the calculation of outliers for processes involving a parameter termed as *delta*. The random selection of test cases is done with the help of `random` package, and the data used for this comes from the high-level processes since we use their duration for *delta* calculation. This *delta* is calculated as the difference in the duration of a process between its pre and post-stages. If the *delta* exceeds a predefined threshold, the process requires further investigation. The investigation involves determining whether the observed *delta* is caused by an outlier or if it signifies a genuine issue.

Given the hierarchical structure of the processes, divided into high-level and low-level categories, the data for low-level processes can be extremely large, reaching up to 100,000 entries. To manage this scale, random testing was employed primarily for the utility that calculates *delta*. This approach allowed for periodic checks to ensure that the processes flagged for investigation were appropriate and aligned with expectations.

The data used for random testing was the high-level process data. In total, we had data from 6 software change deployment qualifications, and each qualification had a pre and post-run, and each run further had 2000 duration values. We performed testing for all six software change deployment qualifications[Sen07]. Although we performed testing for all six qualifications, the process wasn't computationally too expensive. Random testing was implemented in a more limited scope and serves as an add-on to ensure that the flagged processes meet the criteria for investigation.

6.2.1 Results and Discussion

Our testing implementation performed successfully for all six qualifications. Furthermore, during testing we found that for one of the qualification a process was marked as OK during manual analysis, however, while testing using our utility, the process was highlighted and flagged for having a *delta* which meant that the manual analysis was completely faulty.

6.3 Test Suite

As mentioned earlier, with the growing number of machine configurations in ASML, trying to automate different steps of the analysis process is of key importance, but another aspect that we need to focus on is that for every different machine configuration, the parameters of the utility developed in our study need to be updated as well. However, since the utility follows a hierarchical approach and there are many different steps in the utility, testing each step after updating the parameters can become extremely time-consuming. To solve this, we implement the `Test Suite` framework provided by `Python's unittest` library. A test suite is a collection of various test cases that are intended to test a behavior or set of behaviors of a software application or system. Grouping tests into test suites helps manage, execute, and report test results efficiently.

Effectively acting as a container for these test cases, the suite showcases precise details and objectives for each individual test case. Each test case within a test suite checks a specific functionality or a set of functionalities, ensuring software behaves as expected under various conditions. The primary purpose of a test suite is to provide a structured and systematic approach to testing, making it easier to manage, execute, and track the testing process. Furthermore, it includes vital information regarding the system configuration necessary for the testing process. What sets it apart is its utilization of distinct stages to denote the ongoing test execution status—ranging from Active and In-progress to Completed [Lam25].

We included all the testing functionalities that we developed using input space partitioning and random testing into the test suite and finally finished the testing part of the study.

6.3.1 Results and Discussion

Using the Test suite helped us gain better insight into the utility, as we were able to test the connection between the different functionalities present in the utility, which was not possible by testing them individually [MMH21].

7 Conclusion

The ever-growing complexity of lithography machines and the hierarchical nature of their operational processes necessitate innovative approaches to performance evaluation. This thesis proposed a robust framework for anomaly detection in non-stationary duration data, focusing on distinguishing genuine performance issues from outliers. By employing advanced machine learning techniques such as Isolation Forest, One-Class SVM, and Local Outlier Factor, alongside the statistical capabilities of Pynomaly, the proposed system achieves high accuracy, precision, recall, and F1 scores.

The integration of PyNomaly and Isolation Forest successfully addressed the challenges of identifying anomalies in unlabeled datasets. By leveraging the strengths of probabilistic methods and scalable machine learning algorithms, the utility demonstrated strong performance in accurately detecting outliers across both high- and low-level processes. The integration of

hyperparameter tuning, systematic testing, and input space partitioning further enhances the robustness and reliability of the system. These methodologies ensure that the models generalize well to diverse datasets, making them applicable to a wide range of industrial scenarios. Additionally, random testing provides supplementary validation, ensuring the scalability and efficiency of the anomaly detection framework.

Additionally, the preprocessing pipeline ensured data consistency and prepared the dataset for modeling. Through steps like handling missing values, rounding insignificant values, and mapping appropriate data types, the utility was optimized for analyzing lithography machine performance. The hierarchical segmentation of process data provided deeper insights into performance trends, further enhancing the interpretability of the results.

While this study laid the groundwork for an efficient and interpretable anomaly detection framework, future research could explore several avenues for improvement. For instance, incorporating ensemble learning models or advanced preprocessing techniques could further enhance detection accuracy. Additionally, extending the utility to handle real-time data streams would make it more adaptable to dynamic operational environments.

In conclusion, this study offers a reliable and scalable solution for performance evaluation in lithography machines, reducing the time and effort required by engineers and contributing to the broader goal of maintaining high manufacturing quality in the semiconductor industry.

By automating anomaly detection, our framework aims to reduce the time engineers spend on manual performance analysis, allowing them to focus on optimizing machine operations. However, deploying it in a real-time industrial setting at ASML would require addressing computational constraints and ensuring seamless integration with existing analyzer systems. Naturally, beyond ASML, this framework could be adapted for anomaly detection in other semiconductor manufacturing processes.

Our work has certain limitations. The dataset used primarily represents a subset of ASML lithography machines, which may limit generalizability. Additionally, the reliance on unsupervised methods means that model interpretability remains a challenge. Finally, we used GridSearchCV for hyperparameter tuning, but alternative approaches, such as Bayesian Optimization, could be explored and compared for more efficient tuning in future work.

Other future work directions include incorporating datasets from multiple facilities to enhance robustness or exploring more advanced preprocessing techniques, such as adaptive filtering methods, automated features selection, or advanced imputation methods, to further refine and improve the dataset before model training. Finally, one could investigate deep learning-based anomaly detection methods, such as recurrent neural networks (RNNs) or variational autoencoders (VAEs), to improve detection accuracy in highly dynamic environments.

8 Appendix Grid Search for Optimal Hyperparameters for Models

8.1 Local Outlier Factor

1. n_neighbors : [10, 5, 3]
2. contamination : [auto, 0.01, 0.05, 0.1]
3. algorithm : [auto, ball_tree, kd_tree, brute]
4. leaf_size : [5,10,30,50,100]

8.2 One-Class SVM

- nu : [0.01, 0.05, 0.1, 0.5]
- kernel : [rbf, linear, poly, sigmoid]
- gamma : [auto, scale, 0.001, 0.01, 0.1]

8.3 Isolation Forest

1. n_estimators : [100, 50, 200, 500]
2. max_samples : [10, 5, 3]
3. contamination : [auto, 0.01, 0.05, 0.1]

8.4 PyNomaly

1. extent : [1,2,3]
2. n_neighbors : [10, 5, 3]

References

[AASM20] Omar Alghushairy, Raed Alsini, Terence Soule, and Xiaogang Ma. A review of local outlier factor algorithms for outlier detection in big data streams. *Big Data and Cognitive Computing*, 5(1):1, 2020.

[AO17] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2017.

[APB18] Juozas Auskalnis, Nerijus Paulauskas, and Algirdas Baskys. Application of local outlier factor algorithm to detect anomalies in computer network. *Elektronika ir Elektrotechnika*, 24(3):96–99, 2018.

[ASM25] ASML Holding. Wikipedia, the free encyclopedia, 2025.

[BGCML21] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A Lozano. A review on outlier/anomaly detection in time series data. *ACM computing surveys (CSUR)*, 54(3):1–33, 2021.

[BKNS00] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[BM07] Sabyasachi Basu and Martin Meckesheimer. Automatic outlier detection for time series: an application to sensor data. *Knowledge and Information Systems*, 11:137–154, 2007.

[Con] V Constantinou. Pynomaly: Anomaly detection using local outlier probabilities (loop). j open source softw 2018; 3 (30): 845.

[Con18] Valentino Constantinou. PyNomaly: Anomaly detection using local outlier probabilities (LoOP). *Journal of Open Source Software*, 3(30):845, oct 2018.

[CQS13] Yuting Chen, Jing Qian, and Venkatesh Saligrama. A new one-class svm for anomaly detection. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3567–3571. IEEE, 2013.

[GD12] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: poster and demo track*, 1:59–63, 2012.

[Ham94] Richard Hamlet. Random testing. *Encyclopedia of software Engineering*, 2:971–978, 1994.

[HS13] Maryamsadat Hejazi and Yashwant Prasad Singh. One-class support vector machines approach to anomaly detection. *Applied Artificial Intelligence*, 27(5):351–366, 2013.

[Lam25] Lambdatest. Understanding test suite test case: Examples and best practices, 2025.

[LHTX03] Kun-Lun Li, Hou-Kuan Huang, Sheng-Feng Tian, and Wei Xu. Improving one-class svm for anomaly detection. In *Proceedings of the 2003 international conference on machine learning and cybernetics (IEEE Cat. No. 03EX693)*, volume 5, pages 3077–3081. IEEE, 2003.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[LTZ12] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):1–39, 2012.

[LZS+10] Moisés F Lima, Bruno B Zarpelao, Lucas DH Sampaio, Joel JPC Rodrigues, Taufik Abrao, and Mario Lemes Proença. Anomaly detection using baseline and k-means clustering. In *SoftCOM 2010, 18th International Conference on Software, Telecommunications and Computer Networks*, pages 305–309. IEEE, 2010.

[MMH21] Dominik Meier, Toni Mattis, and Robert Hirschfeld. Toward exploratory understanding of software using test suites. In *Companion Proceedings of the 5th International Conference on the Art, Science, and Engineering of Programming*, pages 60–67, 2021.

[Pan99] Jiantao Pan. Software testing. *Dependable Embedded Systems*, 5(2006):1, 1999.

[PB19] Nerijus Paulauskas and Algirdas Baskys. Application of histogram-based outlier scores to detect computer network anomalies. *Electronics*, 8(11):1251, 2019.

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[SEK05] Hyun Joon Shin, Dong-Hwan Eom, and Sung-Shick Kim. One-class support vector machines—an application in machine fault detection and classification. *Computers & Industrial Engineering*, 48(2):395–408, 2005.

[Sen07] Koushik Sen. Effective random testing of concurrent programs. In *Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering*, pages 323–332, 2007.

[sf] Python software foundation. *Python unittesting library — Unit testing framework*.

[SFAA23] Gerard Shu Fuhnwi, Victoria Adedoyin, and Janet O Agbaje. An empirical internet protocol network intrusion detection using isolation forest and one-class support vector machines. 2023.

[SWS+99] Bernhard Schölkopf, Robert C Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. *Advances in neural information processing systems*, 12, 1999.

[TAP14] Jing Tian, Michael H Azarian, and Michael Pecht. Anomaly detection using self-organizing maps-based k-nearest neighbor algorithm. In *PHM society European conference*, volume 2, 2014.

[Wik24a] Wikipedia. Interquartile range. *Wikipedia*, page 1, 2024.

[Wik24b] Wikipedia contributors. Isolation forest — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Isolation_forest&oldid=1261659037`, 2024. [Online; accessed 29-December-2024].

[Wik24c] Wikipedia contributors. Local outlier factor — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Local_outlier_factor&oldid=1224921028`, 2024. [Online; accessed 29-December-2024].

[WWLT14] Xiang Wan, Wenqian Wang, Jiming Liu, and Tiejun Tong. Estimating the sample mean and standard deviation from the sample size, median, range and/or interquartile range. *BMC medical research methodology*, 14:1–13, 2014.

[ZP17] Chong Zhou and Randy Clinton Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 665–674, 2017.