

Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (37.th cycle)

Computational Intelligence for Computer-Aided Design

Machine Learning Techniques for Microcontrollers Performance Screenings

Nicolo' Bellarmino

* * * * *

Supervisors
Prof. R.Cantoro., Supervisor
Prof. G.Squillero., Supervisor

Politecnico di Torino
December 13, 2024

This thesis is licensed under a Creative Commons License, Attribution - Noncommercial-NoDerivative Works 4.0 International: see www.creativecommons.org. The text may be reproduced for non-commercial purposes, provided that credit is given to the original author.

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Nicolo' Bellarmino

Turin, December 13, 2024

Summary

Microcontrollers are now integral to nearly all electronic applications, ranging from home automation to aerospace systems. In safety-critical domains such as automotive control, ensuring high quality and safety standards necessitates rigorous performance screening. This process identifies underperforming devices by determining the maximum frequency ($F_{\max}$) at which they can operate under challenging conditions, such as extreme temperatures and voltages. Traditionally, this is achieved through speed-binning, a method that categorizes devices into performance bins (e.g., fast or slow). However, speed-binning is both costly and time-consuming. This thesis explores the application of Machine Learning (ML) techniques to enhance the performance screening process of microcontrollers, with a particular focus on identifying malfunctioning devices using on-board embedded ring oscillators known as Speed Monitors (SMONs). The primary objective is to develop a comprehensive ML-based framework for MCU performance screening, addressing various challenges (such as noise in data) while developing predictive regression models for the $F_{\max}$. The framework includes data sampling, pre-processing, device characterization, feature selection, and training optimizations through Active Learning and Deep Learning. Several regression models are evaluated and compared for predicting the maximum operating frequency of the devices. Key techniques include data cleaning and outlier detection, which focus on identifying anomalous samples, and multi-task regression, which leverages multiple testing patterns from the label acquisition phase. Due to the substantial expense of acquiring labeled data for ML models training, an Active Learning approach is employed to optimize the training set, thereby reducing the number of labeled samples required to attain a target prediction accuracy by half. The thesis also investigates semi-supervised learning techniques to utilize unlabeled production data, and transfer learning strategies aimed at minimizing the need for labeled data when developing models for new product families. Feature adaptation is used to adjust models trained on previous product families to fit new SMON sets, while feature selection techniques are applied to identify the most informative SMONs early in the production process. Through this research, the work aims to significantly improve the efficiency and accuracy of microcontroller performance

screening, ultimately contributing to more reliable and cost-effective production processes.

Acknowledgements

I would like to thank my supervisors, Riccardo Cantoro and Giovanni Squillero, for the opportunity to work on these interesting and innovative topics during my PhD. In particular, I would like to thank Riccardo for his guidance, passion, and commitment to hard work, research, and education that he has instilled in me over the last five years of successful collaboration. Hard but great times. I would like to thank Alberto Bosio for his patience and guidance during my months in France. I would also like to thank Infineon Technologies AG [1] for allowing me to work on this project, providing me with the data set and introducing me to the semiconductor manufacturing industry.

[1] https://www.infineon.com/

Come può uno scoglio arginare il mare?

"Il cammino dell'uomo timorato è minacciato da ogni parte dalle iniquità degli esseri egoisti e dalla tirannia degli uomini malvagi. Benedetto sia colui che nel nome della carità e della buona volontà conduce i deboli attraverso la valle delle tenebre; perché egli è in verità il pastore di suo fratello e il ricercatore dei figli smarriti."

Dedicato al viaggio e ai suoi imprevisti, e a ogni ostacolo—tappa verso la scoperta di sé e lanterna sul nostro cammino. Alla luce che ci guida, che illumina senza mai distoglierci da ciò che amiamo. Alle forze trovate nel buio. Chiù scuru di mezzanotti nun po' fari.

Contents

List of Tables xi

List of Figures xiii

1 Introduction 1
- 1.1 Problem Description and Motivations . . . 1
- 1.2 Challenges and Objectives . . . 2
- 1.3 Contribution, Proposed Flow and Organization . . . 4

2 Background: Microcontrollers 7
- 2.1 Microntrollers . . . 7
- 2.2 Testing of Microcontrollers . . . 8
- 2.3 SMON Structures and Their Application in Performance Screening 10
- 2.4 Characterization Using Split Lots and SMONs . . . 11
- 2.5 Performance Evaluation Using SBST and Pattern-Based Testing . . 12

3 Background: Machine Learning 15
- 3.1 Introduction to Machine Learning . . . 15
- 3.2 Active Learning . . . 17
  - 3.2.1 LOF: Local Outlier factor . . . 20
  - 3.2.2 QBC: Query by committee . . . 21
  - 3.2.3 EDD: Euclidean Distance . . . 22
  - 3.2.4 GSx, GSy, iGS . . . 22
  - 3.2.5 Hausdorff Distance . . . 24
- 3.3 Multi Task Learning . . . 24
- 3.4 Dataset Shift and generalization problem . . . 26
- 3.5 Dimensionality Reduction . . . 27
  - 3.5.1 Compressed Sensing . . . 29
- 3.6 Machine Learning model used . . . 29
  - 3.6.1 Linear Models . . . 30
  - 3.6.2 SVM . . . 33

3.6.3 Tree Based Estimator . . . 34
3.6.4 Kernel Ridge Regression . . . 35
3.6.5 Multivariate Adaptive Regression Splines (MARS) . . . 35
3.6.6 Neural Networks . . . 36
3.6.7 Transfer and Semi-Supervised Learning . . . 39
3.7 Model Evaluation . . . 40
3.7.1 Regression . . . 41
3.7.2 R^2 . . . 41
3.7.3 Classification . . . 43
3.8 Outlier Detection . . . 45
3.9 Imputation . . . 47

4 Related Work of Machine Learning Applications in IC Testing and CAD 49
4.1 The state-of-the-art microntroller performances prediction . . . 50

5 ML-based Performance Screening Foundation 53
5.1 Data Collection . . . 53
5.2 Machine Learning models . . . 57
5.2.1 Polynomial Transformation . . . 58
5.2.2 Feature Space reduction . . . 58
5.3 Multi Target learning . . . 59
5.4 Active Learning . . . 59
5.5 Experiments and Results . . . 62
5.5.1 Basic Models . . . 63
5.5.2 Advanced Models . . . 64
5.5.3 Multi-Target Regression . . . 67
5.5.4 Discussion on ML models . . . 70
5.5.5 Active Learning . . . 70
5.5.6 Discussion on Active Learning . . . 72
5.6 Optimize the training set . . . 74
5.6.1 Experiments . . . 77
5.7 Proposed Overall Flow . . . 80
5.8 Conclusion . . . 82

6 Deep Learning in MCU Performance Screening 85
6.1 Motivation . . . 87
6.2 New Products . . . 89
6.3 Proposed Approach . . . 90
6.3.1 Pseudo-Labeling . . . 93
6.3.2 Auto-Encoder . . . 94

6.3.3 Denoising Auto-Encoder . . . 94
6.4 Experimental Evaluation . . . 94
6.4.1 Setup . . . 94
6.4.2 Inter-Family Transfer Learning Setup . . . 98
6.4.3 Deep Learning Models . . . 99
6.4.4 Experimental Results . . . 100
6.5 Limitations . . . 105
6.6 Conclusion . . . 106

7 Feature Selection 109
7.1 Motivation . . . 111
7.2 MCU products . . . 113
7.3 SMONs module . . . 114
7.4 Comparative analysis . . . 114
7.4.1 Wrapper Methods . . . 115
7.5 Experimental Evaluation . . . 116
7.5.1 Experimental Setup . . . 116
7.5.2 Multicollinearity analysis . . . 118
7.5.3 Model metrics: Dataset *A* . . . 121
7.5.4 Model metrics: Dataset *B* . . . 122
7.5.5 Time Analysis . . . 123
7.6 Conclusions . . . 125

8 Conclusion 131
8.1 Future Work . . . 132
8.1.1 Side Activities . . . 134

A List of Published Papers 135

Bibliography 139

List of Tables

3.1 Examples of activation functions commonly used in neural networks. 37
3.2 Examples of loss functions utilized in neural network training. Here, y represents the predicted output of the network, while $\hat{y}$ denotes the true target values, and C is the number of classes. 38
3.3 Example of confusion matrix 44
5.1 Normalized root mean squared error (nRMSE) on the test set using basic models 64
5.2 Basic models scores (mean values on 10 patterns) 65
5.3 Comparison between PCA and UFS (mean nRMSE on 10 patterns, advanced independent models) 68
5.4 Advanced models vs Base models scores (mean values on 10 patterns, independent target) 69
5.5 Pearson correlation between Labels 69
5.6 Comparison between advanced models in-chain or single target (P_{min} target), average over 20 run. 71
5.7 Area under the nRMSE curve for different algorithm (Polynomial Ridge, Random Forest, SVR) 72
5.8 Polynomial Ridge Regression Mean and variance nRMSE during active learning using all possible initial wafer choices 74
5.9 Correlation between selection strategies and nRMSE on the f_{max} Ridge regression models with polynomial features 75
5.10 Prediction performances of P_{min} target on test set with different methods for outlier detection on labels (Polynomial Ridge Regression) 78
6.1 Prediction performance metrics of deep feature extractors on product *A1* 102
6.2 Prediction Performance Metrics of Intra-Family Transfer Learning on Product *A2* 103
6.3 Average Results on $B1$ for 5 Different Training-Test Splits. *Best Feature Selection (B-FS) is considered a "gold standard" for comparison, as the importance of each SMON was determined using the entire labeled set. 107

7.1 R^2 Score (%) on datasets A and B with RandomForest and different feature sets (5-splits) 119
7.2 Error metrics on A, 5-folds average, divided by feature set (in bold) 123
7.3 Error metrics on B, 5-folds average, divided by feature set (in bold) 124

List of Figures

2.1 Data Collection steps through manufacturing 11
3.1 Example of a ResNet18 neural network applied to a binary classification problem with image inputs. The network outputs the probabilities associated with each class. 37
5.1 An overview of functional patterns and their critical fail frequency with the threshold frequency level of the data-sheet. Some outliers are visible over all applied patterns [44]. The threshold is 0%. . . . 55
5.2 Graphical view of a toy-example of label measurements for a device. Five patterns are measured and five measurements are taken for each pattern. In this example, the frequency for patterns 2 and 3 would be discarded from the dataset because of the high variance. . 56
5.3 Two-dimensional PCA representation of corner-lot devices used for initial model characterization. The color scale indicates the actual frequency of the devices in Hz relative to the specification f_{req}. . . . 60
5.4 Graphical illustration of Regressor Chain (above) and 2-Stage Regressor (below) training and prediction procedures [130] 61
5.5 PCA-based two-dimensional representation of the corner-lot devices used for the initial model's characterization, colored by wafer. . . . 64
5.6 Spearman Rank Correlation matrices for product A, with 27 SMONs. Clusters of features can be identified. 66
5.7 nRMSE obtained with different regression algorithms, using AL strategies. The curves represent the mean value of the errors depending on all possible initial wafer choices. 73
5.8 Correlation between active learning unsupervised scores (EDD, LOF, QBC, GSx, GSy, iGS, HD) and nRMSE using polynomial Ridge regression. 75
5.9 Steps performed during the data preparation phase on the labels dataset. 76

5.10 Learning curves of Ridge (upper plot) and Kernel Ridge (lower plot) models with different types of training sets, tested on the same test set. The x-axis represents the number of devices characterized (constant), while the y-axis indicates the nRMSE. The dashed lines represent the nRMSE values achieved by the baseline models (1, red) with the maximum number of samples in the training sets. . . 81
5.11 Proposed performance screening flow 82
6.1 PCA-based two-dimensional representation of the corner-lot devices used for the initial model's characterization, colored by wafer, product $A2$ 90
6.2 PCA-based two-dimensional representation of the corner-lot devices used for the initial model's characterization, colored by wafer, product B. Here the distinction among wafer is less marked. 91
6.3 The proposed Inter and Intra-Family transfer learning protocol. . . 92
6.4 An example of training a deep convolutional feature extractor in a Semi-Supervised fashion using the Auto-Encoder alternate task. Initially, the deep model is trained with unlabeled data, and then it is used as a transformer for labeled data (for the same or a different product). Finally, a Linear Model is trained to link the feature embeddings with the labels. 95
6.5 An illustration of the proposed transfer learning approach between the *A1* and *A2* products using the PL-CNN. The *A2* dataset is first transformed using the pre-trained network, which remains frozen, with no updates to the convolutional layers. A linear regression model is then trained to link the feature embeddings from *A2* with the $F_{\max}$ labels. 96
6.6 Learning curves for different deep and shallow models. The x-axis represents the number of training samples (log scale, truncated at 400 samples), and the y-axis represents the nRMSE on the test set. The upper dotted line denotes 2% nRMSE, while the lower line indicates the baseline Polynomial Ridge prediction error obtained with the entire training set (1.57% nRMSE). 102
6.7 Learning curves for different models on the *A2* product. The x-axis represents the number of training samples (log scale), while the y-axis represents the nRMSE computed on the test set. The upper dotted horizontal line corresponds to 2% nRMSE, and the lower horizontal line indicates 1.54% nRMSE, which is the final error obtained by the Polynomial Ridge. The use of Transfer Learning and Deep Feature Extractors allows for lower prediction error with a smaller training set. 104

6.8 Learning curves for different models on product $B1$. The x-axis represents the number of training samples (log scale) while the y-axis shows the nRMSE computed on the test set. The upper dotted horizontal line corresponds to 2% nRMSE, and the lower horizontal line represents 1.52% nRMSE, the final error achieved by the Polynomial Ridge model. Model simplification (via FS) and the application of Transfer Learning reduce the number of labeled samples required for model training. 106
7.1 Spearman Rank Correlation matrices for two products: A, with 27 SMONs (left) and B, with 579 SMONs (right). In B, SMONs are highly correlated. In A, clusters of features can be identified. 113
7.2 The Dendograms, results for the hierarchical clustering procedure, for dataset A (upper) and B (lower). By visually inspecting the dendograms, it is possible to set a threshold, to cut it at a certain level, extracting a certain number of cluster. On the y-axis, the distances between clusters. On the x-axis, the SMONs that form each (colored) cluster 120
7.3 Permutation Importance Index for dataset A 126
7.4 Permutation Importance Index for dataset B 127
7.5 Learning curves for different models on A dataset. The x-axes is the number of training samples (log scale) while the y-axes is the nRMSE computed on the test set. The upper plot shows the overfitting of Linear Regression and the Shark-tooth behavior of $\ell_{\frac{1}{2}}$. We disabled that traces in the lower plot. CorrT techniques collapse to 1.94% nRMSE. All the other models are aligned in terms of final error metrics, with slightly better results in terms of error/coefficient for the non-convex LGP model. 128
7.6 Learning curves for different models on dataset B. The x-axes is the number of training samples (log scale) while the y-axes is the nRMSE computed on the test set. The upper plot shows, even for this dataset, the overfitting of Linear Regression and the Shark-tooth behavior of $\ell_{\frac{1}{2}}$. We disabled the traces in the lower plot. LASSO model is the most able to reach good balance between number of feature chosen (60) and prediction accuracy (0.89% nRMSE). Non-convex techniques tend to prefer sparser solutions, at the cost of an increase in the prediction error. 129

Chapter 1

Introduction

1.1 Problem Description and Motivations

In safety-critical industries such as automotive and aerospace, the reliable performance of microcontrollers (MCUs) is paramount. Ensuring the dependability of these devices is crucial, particularly when they are integrated into critical components. To verify the performance of electronic devices in the automotive and aerospace industries, traditional techniques have typically employed structural tests such as transition delay testing and path delay testing, functional tests, or other indirect monitors [55, 158, 50, 51, 102, 142, 47]. Traditionally, the screening of MCUs has focused on identifying devices that meet specific criteria, with a significant emphasis on the maximum operating frequency ($F_{\max}$). Performance screening aims to precisely determine $F_{\max}$ of a device under various worst-case conditions. This conventional approach involves extensive testing at increasing clock frequencies, a process that is not only time-consuming but also relies on expensive test setups. Additionally, these methods typically provide categorical results, which may not offer the nuanced information needed for optimal performance assessment.

Machine Learning (ML), a dynamic field within the broader realm of artificial intelligence, has continuously evolved over the years, driven by the immense interest of researchers. ML is widely used to support decision-making processes across various domains by providing predictions based on models that automatically extract knowledge from data. This capability of ML to learn and make predictions autonomously makes it an invaluable tool in enhancing the efficiency and effectiveness of performance screening in the automotive and aerospace industries.

In recent years, ML regression models have been proposed as a solution to predict the $F_{\max}$ of MCUs based on correlated data, offering substantial time savings compared to traditional methods [44].

For instance, oscillation frequency values from Ring Oscillators (ROs) can be utilized to predict $F_{\max}$ across different test scenarios (*labels*) using ML models. This technique is not intended to replace structural testing but rather to complement it by providing more informative results compared to simple speed binning, which sorts devices into categorical bins like "fast" and "slow". [55, 158, 50, 51, 102, 142, 47].

The *labels*, thus, the MCU $F_{\max}$ frequencies, are acquired by applying functional test patterns that replicate customer use cases of the MCU. This characterization cannot be done directly in production, but necessitates a separate step during the engineering phase, and therefore requires significant time and resources.

ML-based models for performance screening offer substantial benefits to test and product engineers: the response of these models is continuous rather than categorical, thus increasing the information extracted by the performance screening. Once the model is built and working, it can be deployed in production, possibly enabling large-scale performance screening regression. This can be seen as an additional screening test that comes with no time cost for silicon producers. By leveraging ML models to predict $F_{\max}$, it is possible to streamline the screening process, reduce reliance on expensive testing setups, and achieve more detailed performance insights. This integration of ML techniques represents a significant advancement in the methodology for evaluating the reliability and performance of MCUs in safety-critical applications.

The goal of this thesis is to provide a fundamental and comprehensive framework for ML-based MCU performance screening from a data analysis perspective. This involves developing methodologies that not only enhance the accuracy in predicting the maximum operating frequency ($F_{\max}$) but also streamline the entire screening process. By leveraging ML techniques, the framework aims to deliver more nuanced insights compared to traditional methods, facilitating improved decision-making in safety-critical industries such as automotive and aerospace. The thesis aims to show the principal challenges in building ML-based performance screening systems and ways to address these.

1.2 Challenges and Objectives

While ML-based techniques can offer substantial benefits when put into production, several challenges have to be addressed. A crucial factor in the accuracy of supervised ML models is the availability of high-quality labeled data. Unfortunately, collecting $F_{\max}$ data is a laborious task, and it can be challenging. The number of labeled data available for the learning phase is often small, ranging from hundreds to thousands of devices. Constructing a new ML model from scratch under such circumstances is time-consuming and not always feasible.

Process variations can significantly impact the training phase of a prediction model, leading to biased or unstable results. Process variation in silicon manufacturing refers to deviations in fabrication parameters from their nominal values, caused by factors like equipment limitations, environmental conditions, and material properties. It is categorized as:

- **Systematic variations**: Predictable deviations, e.g., lithography or etching inconsistencies.
- **Random variations**: Unpredictable deviations, e.g., atomic-scale defects or dopant fluctuations.

These variations impact device parameters (e.g., threshold voltage, channel length) and circuit performance (e.g., speed, power, reliability). If the training data is collected during periods of high process variation, ML models may learn patterns that are specific to those conditions rather than the true underlying relationships between *features* and *labels* used as input. This can lead to biased predictions when models are applied to new data with different variations. Thus, high process variation lead to noisy training data. If the model is too complex, it may overfit to this noise, capturing the random fluctuations rather than the true patterns. This results in a model that performs well on the training data but generalizes poorly to new situations. Process variations can cause the model performance to vary significantly depending on the specific subset of training data used. A model with high variance will be very sensitive to the training data and may produce inconsistent predictions when trained on different batches. Strategies for choosing the best possible initial training set need to be implemented, in addition to data cleaning techniques and outlier removals.

Accurate estimation of model error is indeed essential for ensuring the reliability and robustness of predictions made by ML models. A well-quantified model error allows practitioners to understand the model's performance, identify areas for improvement, and make informed decisions based on the predictions.

In the rapidly evolving field of microelectronics, new product releases are frequent. Adapting the existing ML models to accommodate new products is a significant challenge. As new products are introduced, the underlying data distribution changes, and the ML models need to be updated to maintain their predictive performance. Traditional ML models are typically trained on a fixed dataset and do not adapt well to changes in the data distribution over time.

Data comes from production stages that can be non-stationary environments, changing over time (i.e, our models may become no accurate anymore). Also, production present inter-wafer process variation that can lead to loss of accuracy in predicting F_{max}. Using a pre-existing model trained on previous-generation data

may not yield accurate results due to shifts in data distributions, introducing the need to re-train or fine-tune the model.

The performance of each MCU is measured by running several different labels (or Patterns, section 2.5) at increasing $F_{\max}$ until a failure is recorded. Thus, multiple labels are available in the dataset. To be more conservative, the final target is the critical patterns, or the minimum maximum operating frequency among all the patterns. Even if the final models consist of a single output, using all the possible labels can give more insights about the performance of each devices.

This work aims to address these challenges by optimizing the ML model training process, reducing the need for a large number of labeled samples, minimizing prediction errors, and defining a framework to build predictive models that can be re-used among product families.

1.3 Contribution, Proposed Flow and Organization

This thesis provides a thorough examination of an ML-based framework for microcontroller performance screening, from the initial stages of data acquisition to its deployment in the field.

The primary contributions of this work are outlined as follows:

- Documentation of the data acquisition and preprocessing methodologies applied to split-lot data.
- Development and training of preliminary models based on the preprocessed data.
- Evaluation of these models to estimate potential prediction errors.
- Extension of the modeling approach to new data and products, potentially involving different SMON sets.
- Optimization of the characterization process by minimizing the quantity of labeled data required for model training.
- Utilization of unlabeled production line data to enhance model robustness.
- Development of models that are transferable across different product families.
- Guidance on selecting relevant ROs to be integrated into new products, thereby influencing design decisions through data-driven insights.

The structure of this thesis is organized as follows: Chapter 2 provides a comprehensive overview of the industrial processes and technologies associated with microcontrollers, establishing the necessary context for the study. Chapter 3 delves into the Machine Learning concepts pertinent to this research, laying the theoretical groundwork for the subsequent chapters.

In Chapter 4, a concise review of the existing literature on performance screening is presented, serving as the baseline knowledge and foundation for the new experiments conducted in this work.

Subsequent chapters (Chapters 5 to 7) details the investigations and methodologies developed for the proposed approach.

Chapter 5 introduces the methodology employed in this research, along with preliminary experiments focused on model development, multi-target learning, and active learning. It delves into the initial phase of developing ML models utilizing split-lot data. In semiconductor manufacturing, split-lots are instrumental in analyzing process variations and enhancing operational efficiency. The data derived from these splits serve as a valuable resource for identifying features that exhibit strong correlations with $F_{\max}$. The preliminary modeling phase involves thorough data preprocessing to mitigate measurement errors and exclude outliers in $F_{\max}$ values. Upon cleaning and refining the dataset, ML models are trained to establish a mapping function $y = f(x)$, translating SMON frequencies into $F_{\max}$ performance predictions.

The resulting models serve as the foundational elements of the performance screening framework, providing a critical baseline for subsequent experimentation.

In the proposed framework, model predictions are then compared against a predetermined performance threshold, categorizing devices as either pass or fail. The accuracy of these predictions is vital and is typically evaluated using techniques such as cross-validation, which helps identify issues like overfitting or underfitting. Additionally, methods like bootstrapping offer further insight into the variability and robustness of model outcomes. Reliable error estimation is essential for confidence in the ML-based screening process, which necessitates the implementation of a risk-based *Guardband* [150]. This buffer surrounds the specified limits to ensure that minor variations or uncertainties do not compromise product quality or reliability.

As new products are introduced, the relationship between SMONs and $F_{\max}$ may evolve due to changes in process parameters or MCU specifications. New models must be developed, either from the ground up or by leveraging existing models trained on previous product generations.

Building upon this foundation, Chapter 6 explores the application of continuous learning techniques, incorporating deep learning, semi-supervised, and transfer learning methodologies. These approaches enable the models to be updated

dynamically as new data becomes available, ensuring adaptability to new product releases and manufacturing conditions. Furthermore, the optimization of the characterization phase and the reduction of training set size are facilitated by deep learning models trained on production line data, which are able to reduce the effort in labelling devices up to six times. Transfer learning, in particular, allows for the adaptation of existing models to new products with minimal additional data (in the order of units/tens of samples), reducing both training time and data requirements, and supporting continuous model refinement as more data is accrued.

Chapter 7 discusses the methodology for selecting the most effective SMONs as indicators of device performance, highlighting the critical role of feature selection in the overall framework. When designing new products, a preliminary analysis of SMONs is crucial for selecting the most effective set for performance screening. Consequently, feature selection methods become essential. The chapter presents various feature selection strategies and compressed sensing techniques, guiding design decisions regarding the most appropriate ROs to incorporate for optimal performance screening.

Finally, Chapter 8 offers a concise summary of the key findings and conclusions derived from the research, encapsulating the main contributions and implications of the work.

Chapter 2

Background: Microcontrollers

2.1 Microntrollers

Microcontrollers are integral components in modern electronic systems, serving as compact, programmable devices that control various functions in embedded systems. Here's an overview of what microcontrollers are, their purposes, how they are created, fabricated, and tested.

A microcontroller [124, 59] (often abbreviated as MCU or μC) is a small computer on a single integrated circuit (IC) designed to manage specific tasks within electronic devices. Microcontrollers typically include a processor core, memory (both volatile and non-volatile), and various input/output peripherals. They are primarily used in embedded applications, where they execute pre-defined instructions to control devices such as home appliances, automotive systems, medical devices, and consumer electronics. This is often done without the interceding of a complex front-end operating system, but programming directly the MC in a bare-metal fashion, either in high-level programming language (C, $C++$) or directly on instruction level Assembly language, in order to perform one task repeatedly. Microcontrollers are designed to perform dedicated functions. Their compact size and low power consumption make them ideal for applications requiring control, automation, and data processing. The creation of a microcontroller begins with the design phase, where engineers define the architecture, including the CPU, memory types (RAM, ROM, EEPROM), and peripheral interfaces. This phase involves using hardware description languages (HDLs) to create the logic circuits that will be integrated into the chip. Microcontrollers are fabricated using semiconductor manufacturing processes, which typically include:

- **Wafer Fabrication**: Silicon wafers are processed through photolithography, etching, and doping to create the various components of the microcontroller, such as transistors, resistors, and capacitors.

- **Packaging**: After fabrication, the silicon chips are cut from the wafer and packaged into protective casings that allow for easy integration into electronic circuits.
- **Testing**: Each microcontroller undergoes rigorous testing to ensure functionality, performance, and reliability. This includes electrical testing, functional testing, and stress testing to identify any defects or failures.

2.2 Testing of Microcontrollers

Test activities are conducted at various stages throughout the life cycle of an integrated circuit (IC) to ensure that the final product meets its specifications and is reliable for market use. These activities encompass several phases, each with distinct objectives and methodologies. Initial validation tests are performed during the design phase to ensure the product meets its specifications. These tests are crucial for identifying and correcting design flaws early in the development process. Characterization tests are conducted on the first prototypes to identify any failures or faults. These tests may necessitate a redesign of the product if significant issues are discovered. Characterization helps in understanding the device's behavior under various conditions and guides necessary adjustments before moving to large-volume production. Once the device passes characterization tests, it enters large-volume production. During this phase, chips are tested directly on-wafer to detect defects before they are diced and packaged. Post-packaging tests confirm that the packaging process has not introduced new faults and that the ICs are ready for market release. After market release, in-the-field tests are conducted to ensure the IC performs correctly in specific applications. These tests help in identifying any issues that may arise under actual operating conditions and can guide further improvements or adjustments. Testing ICs is both time-consuming and expensive, with costs rising rapidly with the level of integration. Functional tests are used to verify that the IC meets its specifications. However, the complexity of modern digital microcontrollers makes exhaustive functionality verification impractical ([116]). Structural tests focus on identifying faults in the circuits rather than verifying the functionality of each individual part. This approach often requires costly external test equipment due to the diversity and pin counts of chips. To reduce costs, test features can be integrated into the chip. One strategy is to include dedicated structures within the circuit, known as Design-for-Testability (DfT), which simplifies the development and application of test procedures. This structural paradigm ensures correct circuit manufacturing, but designing and inserting these additional testing structures can increase design time and consume physical space on the die. For MCUs or System-on-Chips (SoCs) that include

at least one microprocessor, suitable test programs can be executed by the processor to test the device itself. These programs stimulate various parts of the circuit, and the responses are collected for evaluation. This strategy, known as Software-Based Self-Test (SBST), offers multiple advantages: it is non-intrusive, does not require extra hardware, and tests can be performed at the processor's operating frequency. This enables the detection of defects not noticeable at lower speeds, prevents overtesting, and can be applied in the field during the product's lifetime. Additionally, these tests do not require expensive test equipment and do not involve changes to the device design ([107]).

Performance Screening Manufacturers must assure their customers that the delivered MCUs fulfill the specifications outlined in the datasheet. Among these, there is the maximum operating frequency. Performance screening involves testing the devices and discarding those that do not satisfy constraints on the maximum operating frequency under critical conditions of temperature and voltage and during the execution of different tasks. The desired level of confidence varies depending on the application; devices used in safety-critical applications demand significantly higher effort in performance screening compared to those intended for the low-cost consumer market. Not all devices pass performance screening due to issues such as defective high-ohmic contacts or variations in physical parameters from chip to chip. Therefore, a precise strategy to identify faulty devices is essential. Various approaches exist for identifying faulty circuits. Among these, there is the frequency stepping. Frequency stepping is a technique used in MCU performance screening to identify faults or weaknesses by varying the clock frequency during testing. The main steps involved in this approach are:

- **Initial Setup:** The MCU is configured to run at its nominal operating frequency.
- **Stepping Through Frequencies:** The clock frequency is incrementally increased or decreased in small steps. This means running the MCU at various frequencies within its operating range.
- **Monitoring Performance:** At each frequency step, the MCU's performance and behavior are monitored. This includes checking for correct operation, timing issues, and any anomalies.
- **Detection of Faults:** Faulty circuits may exhibit instability or errors at certain frequencies, which can be detected during the stepping process. If the MCU fails or behaves unpredictably at specific frequencies, it may indicate issues such as timing problems, signal integrity issues, or design flaws.
- **Analysis:** The results from the frequency stepping test are analyzed to

determine if the MCU operates correctly across the entire frequency range. This helps in verifying the robustness and reliability of the MCU design.

Frequency stepping is particularly useful for identifying issues related to clock domain crossing, timing margins, and overall circuit robustness, making it a valuable technique in MCU performance screening and quality assurance. Additional information regarding the deployed frequency stepping framework and the labeling acquisition procedure is presented in section 2.5.

2.3 SMON Structures and Their Application in Performance Screening

The SMONs consist of library cells arranged in series, which exhibit an overall inverting behavior, resulting in oscillation. The number of SMONs can vary by product, ranging from tens to hundreds. Typically, the majority of SMONs are concentrated in a compact block at the die center, while additional SMONs are distributed across the chip to account for spatial variations, commonly referred to as *global mismatch*.

The SMONs on the die are categorized into five distinct groups based on their structure: INV, NAND, NOR, EXT, and VM. These are detailed as follows:

- **INV-ROs**: These SMONs use inverters from the standard library. This symmetric cell type reveals the behavior of both n-MOSFETs and p-MOSFETs in a similar manner.
- **NAND SMONs**: These create an n-biased behavior due to the presence of two daisy-chained n-MOSFETs, which dominate the NAND cell's behavior.
- **NOR-ROs**: These demonstrate a p-biased behavior. This behavior is further reinforced by the placement of dedicated transistors [6].
- **EXT SMONs**: These are more sophisticated. A netlist of the cell design is retrieved, and a specific path is extracted and replicated as an RO. EXT ROs behave similarly to the selected path from the design.
- **VM-ROs**: These are primarily designed and placed to monitor physical parameter variations across the chip.

All five groups of SMONs are manufactured using two main types of transistors: Regular Threshold Voltage (RegVT) and High Threshold Voltage (HVT). RegVT transistors are characterized by high performance but contribute significantly to

leakage current due to their lower threshold voltage. In contrast, HVT transistors exhibit slower switching behavior but substantially reduce leakage current.

Depending on the MCU specifications—whether oriented towards high performance or low power—designers and EDA tools must choose between HVT or RegVT transistors. Consequently, the final design typically includes a mixture of both types. Some paths in the design consist of transistors with a uniform threshold voltage, while others incorporate a mix of HVT and RegVT transistors. As a result, some SMONs are manufactured with uniform threshold voltage transistors, while others feature a combination of both types.

The SMONs are measured in an early production phase when the dies are still on the wafer. The wafer is mounted on a chuck which has high-temperature stability during the measurement process. An internal counter measures the oscillation frequencies of the SMONs. Previous to measurement, each SMON is calibrated to ensure high quality. This calibration is implemented in the software for the measurement process. The stable temperature and calibrated measurement generates high-precision SMON data. The average error is less than 0.15% at the minimum voltage V_{min}. All SMONs data are logged for further processing. The SMON readout can be integrated into the test program of the production test flow. Thus, the SMON data is stored for every produced MCU, possibly leading to millions of unlabeled samples. The SMONs data are acquired both before and after packaging the dies, to analyze the effect of burn-in on the features. In fig. 2.1, the manufacturing procedure and data acquisition steps are represented.

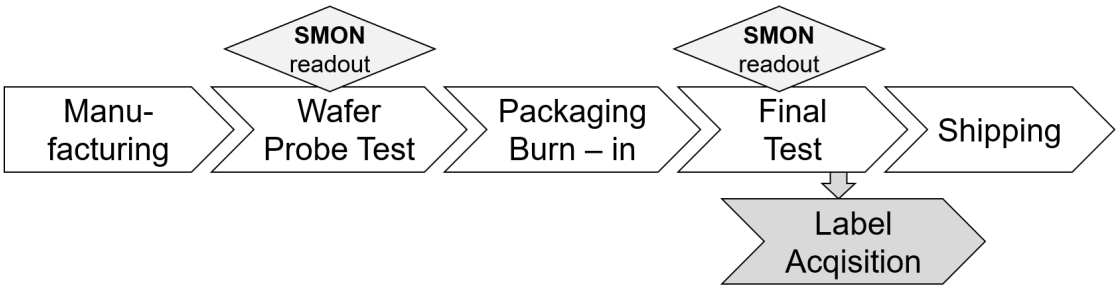


Figure 2.1: Data Collection steps through manufacturing

2.4 Characterization Using Split Lots and SMONs

Split lots refer to a set of wafers produced solely for engineering and analytical purposes. These wafers are fabricated but are not intended for customer delivery. The primary use of split lots is to explore the parameter space, investigate corner cases, and experiment with design parameters to achieve an optimal trade-off among them.

The objective is to construct a representative set of devices with parameters that reflect those encountered in production. Split lots must balance economic feasibility with timing constraints. Additionally, it is possible to approximate the distribution of device performance in production using a Gaussian distribution. While most devices will meet the performance specifications, not all will. Identifying faulty devices is crucial for predicting operational frequency and detecting defective units in production. The primary focus is on precise performance screening of the slower devices located in the lower tail of the performance distribution.

The combination of diverse SMON types with various split-lot wafers provides a robust foundation for examining chip behavior across all operational corners. This approach significantly enhances the efficacy of data processing and performance analysis.

Training ML models on split-lots, especially designed to include diverse and corner cases, helps ensure the model learns to handle rare and extreme scenarios effectively. This approach improves robustness by providing the model exposure to edge cases that may not occur frequently in typical data, enhancing its generalization to real-world situations.

2.5 Performance Evaluation Using SBST and Pattern-Based Testing

Measuring the reference performance is an elaborate process. Following an SBST approach, several programs must be executed on the microcontroller to collect performance labels, specifically the operating frequency. This process involves applying a series of *testing patterns* to the device. After packaging the die, the maximum operating frequency is determined using a customer application-oriented board.

This pseudo-in-system test board has a high-accuracy radio frequency (RF) socket with low resistance and the least possible parasitics. Furthermore, the test board has high power integrity and on-board energy-storage capacitance to prevent voltage droop during the testing. During the test, the MCU's temperature and voltage can be precisely controlled and are continuously sampled by on-chip voltage and temperature sensors.

The MCU is placed on the RF socket with a semiconductor test handling system to improve the repeatability of the measurement process. The voltage and temperature are adjusted to defined voltage V_{crit} and temperature T_{crit} conditions.

When the voltage and temperature conditions are on the defined steady-state values, the MCU is flashed with the firmware.

Various functional patterns are uploaded and executed on the microcontroller.

Each pattern is designed to simulate different types of customer applications, with some representing complete applications and others focusing on specific subcircuits of the chip, thereby addressing all critical aspects of the device.

The applied functional patterns have a high diversity in stressing various design units of the MCU. The functional patterns are designed so that no customer application fails earlier than one out of the functional pattern. Therefore, some patterns are designed to exhaust the maximal load of the MCU. Those patterns create high load jumps and generate cross-noise between the functional blocks. Other functional patterns stress dedicated critical paths in the design. Thus, all functional patterns attempt to push the MCU to its load limit.

The MCU starts to execute a functional pattern with a low frequency. The operating frequency is then incrementally increased until a failure is observed. This procedure is repeated five times for each pattern, and the median frequency from these measurements is calculated and recorded in a database. In this study, ten distinct patterns were utilized. For each MCU, the most critical pattern is the one with the lowest $F_{\max}$. Thus, for each MCU, this will be the target performance and thus, the associated label. We will call this $P_{\min}$ or *PERF*. If the maximum frequency of this critical pattern falls below a specified performance threshold, the device is classified as defective and subsequently discarded.

However, there is some $F_{\max}$ uncertainty remaining caused by the diversity of potential customer applications. The described labeling and characterization process is a time-consuming procedure performed mostly manually. The labels might be affected by uncertainty. Minor changes in measurement conditions have a high impact. Also, some uncertainty may be introduced by mechanical vibrations, contact resistance (MCU to board), and white noise.

Chapter 3

Background: Machine Learning

3.1 Introduction to Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) focused on developing algorithms that enable computers to learn from data and improve their performance autonomously through experience. The core concept of ML is to train machines to perform cognitive tasks, with the goal of either replacing human decision-making or automating complex analytical processes. The term 'Machine Learning' was introduced by Arthur Samuel in 1959, in a paper describing a checkers-playing program [119]. However, the origins of this field can be traced back to 1950 with Alan Turing's seminal paper on computing machinery and intelligence [139]. Today, ML encompasses various automated processes that enhance their performance over time through experience [78].

An ML model is trained using a set of pre-existing examples, known as the 'training set', to make predictions about new, unseen scenarios or data. The term 'Machine Learning' is frequently used in mainstream media to describe any technique employed for classification or prediction tasks, often conflating it with the broader field of "Artificial Intelligence." Unlike traditional "programming-with-code," which involves applying predefined rules to generate specific outputs, Machine Learning adopts a "programming-with-data" approach, where the machine learns to determine input-output relationships through exposure to numerous training examples.

ML is now prevalent across numerous domains, with significant applications in fields such as computer vision, speech recognition, medical diagnosis, robotics, and autonomous vehicles. Many contemporary ML methodologies are based on established statistical tools, such as logistic regression, which originated from the work of Pierre François Verhulst and Adolphe Quetelet in the mid-19th century [56]. Some techniques date back to the 1950s, like Frank Rosenblatt's Perceptron

[113], a precursor to modern neural networks. Following an initial period of excitement, AI and ML experienced reduced funding and interest during periods known as 'AI winters. However, key theoretical advancements and technological innovations have revitalized the field. Notable milestones include the defeat of chess champion Garry Kasparov by Deep Blue in 1997 [42] and AlexNet's breakthrough performance in the ImageNet competition in 2012 [85].

In a typical ML application, data derived from physical objects and real-world scenarios are analyzed and measured, from which a set of *features* is extracted and fed into an *ML model* to achieve a specific outcome, such as solving a classification or regression task. Features may be numeric (integer or real numbers), categorical, or derived directly from images or video frames. Two common ML tasks are *classification* and *regression*. Classification involves constructing a model to select the correct label from a predefined set, while regression focuses on predicting a continuous numerical value. Both tasks are classified as *supervised* learning, as they rely on a set of known results or targets (the *labels*) for training. The goal of supervised learning is to discover the correct input-output relationship that fits the training data and generalizes to new, unseen data. Additional supervised tasks include ranking and recommendation systems, while *unsupervised* learning models operate without labeled data and aim to uncover data structures, such as clustering, dimensionality reduction, and probabilistic density estimation.

The process of training a model to improve its predictive accuracy is known as *fitting*. In supervised learning, once the model type is selected, its parameters are optimized to achieve the best results on a set of data called the *training set*. This optimization involves minimizing a 'loss' function, which quantifies errors over the training set, using iterative methods like Gradient Descent [115]. The model's performance is then evaluated on a separate *validation set* to assist in model selection and hyperparameter tuning. If the performance is unsatisfactory, alternative models may be considered. To mitigate bias in performance evaluation, *k-fold cross-validation* is employed, where the data is partitioned into k equal-sized subsets. For each subset, it serves as the validation set while the remaining subsets are used for training. The results from these k experiments are averaged, ensuring that each observation is used for both training and validation. This method is widely adopted due to its reliability. Additionally, a *test set* may be used for a final, unbiased performance evaluation.

Balancing model complexity and the size of the training set is crucial for effective supervised learning. *Underfitting* occurs when the model is too simplistic to capture the relationships in the training data, potentially due to insufficient data or overly simple model architecture. Conversely, *overfitting* arises when the model is excessively complex, fitting the training data perfectly but failing to generalize to new data, leading to high error rates on the test set. This issue arises because

the 'Empirical Risk' (the loss function measured over the training set) differs from the 'True Risk' (the probability of misclassifying new data points, typically unknown) [140]. The *learning curve* illustrates performance relative to training set size, helping to identify underfitting and overfitting issues.

Machine Learning models are broadly categorized into *shallow learning* and *deep learning* approaches. Shallow learning algorithms rely on manually crafted features based on heuristics, whereas deep learning methods involve stacking multiple layers of neural networks [100], with the primary distinction being the automatic extraction of features from data (feature learning). For shallow learning, feature selection is a critical step, requiring domain-specific knowledge. *Dimensionality reduction* techniques simplify models by selecting relevant features or constructing new features from the original set. In *feature selection*, an algorithm identifies a subset of the original features, while *feature extraction* creates a new set of features as combinations of the originals.

Choosing an appropriate model involves considerable expertise and balancing various objectives, such as generalizability, the ability to interpret underlying processes, and model explainability. In addition to selecting the model type, hyperparameters must be determined. For instance, in a classification task, one might choose a shallow learning algorithm such as a *decision tree* with specified maximum depth and branching strategy, or a deep neural network with multiple layers and neurons per layer. *Ensemble methods* often yield effective results by combining multiple 'weak learners' to improve performance through techniques such as averaging predictions. The *Random Forest* algorithm, detailed in section 3.6.3, exemplifies a common ensemble method that aggregates results from numerous tree-based weak learners.

3.2 Active Learning

In certain fields, acquiring annotated data presents significant challenges due to the high costs and extended time required for label generation. Frequently, the selection of data for labeling and inclusion in the training set of a machine learning (ML) algorithm is conducted randomly, given that, without domain-specific insights, there is no compelling reason to favor one sample over another. However, such a random selection process can result in suboptimal models that fail to capture the general characteristics of the data domain, leading to limited generalization on unseen data.

To overcome this limitation, a more strategic approach is required for selecting training data that contributes to the development of an optimal model with robust generalization capabilities. Active Learning (AL) is a technique designed

to streamline the data collection process by automatically determining which instances should be labeled to train a model both rapidly and effectively. The fundamental concept of AL involves enabling the algorithm to selectively choose examples for training, rather than relying on random sampling for labeling.

AL operates under the principle that an ML model trained on a judiciously selected, smaller dataset can achieve performance on par with—or even superior to—that of a model trained on a larger, randomly selected dataset [147]. This approach potentially allows for higher accuracy while reducing the number of labeled samples required, demonstrating that "curious" algorithms can be more efficient learners in terms of both the number of samples needed and the amount of knowledge extracted.

In an AL framework, the learning algorithm interacts with a user, querying for labels by proactively selecting examples from a pool of unlabeled data. These queries typically take the form of specific data instances that the algorithm identifies as most informative for improving the model's performance. AL can be effectively applied to various ML models, including Support Vector Machines (SVM), to enhance classification accuracy or regression performance with a limited number of labeled samples. As a substantial portion of time in any ML project is devoted to constructing the appropriate training dataset, AL techniques contribute by ensuring that only the most relevant examples are annotated, thus increasing the efficiency of the overall process.

The efficacy of AL hinges on the principle that the benefit derived from querying a specific label must outweigh the associated cost, based on a predefined score function. There are several methodologies for implementing AL, typically categorized into three main scenarios:

- Stream-based selective sampling [54]
- Pool-based sampling [89]
- Membership query synthesis [157, 123, 9]

In stream-based selective sampling, data samples are presented sequentially, and for each sample, the algorithm determines in real-time whether it is advantageous to request a label or discard the sample. This decision-making process occurs concurrently with the model's training.

Pool-based sampling, the most widely recognized AL scenario, involves the algorithm evaluating the entire dataset to select the most informative samples for labeling. Typically, the active learner is initially trained on a randomly labeled subset of the data, which is then used to identify the instances that would be most beneficial to include in the training set for subsequent iterations of the active learning process.

Membership query synthesis, on the other hand, entails the generation of synthetic data points. In this scenario, the active learner is permitted to generate new instances for labeling, drawing on an underlying natural distribution. This method is particularly suitable for problems where generating data instances is straightforward.

Applications such as Remote Sensing Image Classification [138], Text Classification [135], Optical Networks [12], and microcontroller testing require datasets that accurately represent the underlying domain. Randomized data labeling methods often result in suboptimal training sets, characterized by noisy or redundant data, which can hinder the efficiency of the learning process. To enhance model performance, various methods are available to selectively identify and label the most informative samples for inclusion in the training set. In an AL setting, the selection method used for adding new data points to the labeled training set plays a significant role in the final prediction performance of the overall predictive model, particularly in terms of the number of labeled data points required to reach a certain prediction error.

Several AL settings has beend developed in the past [135, 20, 27, 149, 123, 151] *Active Sampling* methods, where the machine learning model is iteratively updated and used to compute a score function for ranking unlabeled data, require retraining the underlying predictive model at each step. On the contrary, *Passive Sampling* methods, which do not rely on learning a function to estimate the score value of data but instead focus on geometric properties of the data within the feature space. Passive sampling is computationally efficient as it does not necessitate model retraining at each iteration and remains agnostic to the specific task (classification or regression) and the underlying model used.

In this study, a pool-based sampling strategy is adopted, with the primary focus on ranking samples within the unlabeled dataset according to various metrics. The objective is to identify and label the most informative samples based on their distance from the current training set. A total of seven unsupervised metrics are investigated, encompassing both active and passive sampling techniques. The metrics employed in this work include:

- Local Outlier Factor (LOF)
- Query by Committee (QBC)
- Euclidean Distance (EDD)
- Greedy Sampling on the input (GSx)
- Greedy Sampling on the output (GSy)
- Improved Greedy Sampling on both input and output (iGS)

- Hausdorff Distance (*HD*)

By leveraging these metrics, the study aims to identify the most representative samples for labeling, thereby enhancing the efficiency and accuracy of the learning process.

3.2.1 LOF: Local Outlier factor

Outlier detection is traditionally conceptualized as a binary classification task, typically grounded in clustering or density-based algorithms that capture global characteristics of the dataset. These methods often define a hypersphere that globally encompasses the data, thereby distinguishing outliers from "normal" data, where the detected anomalies are considered "global" outliers. In contrast, the Local Outlier Factor (LOF) algorithm [37] emphasizes the local properties of the data by focusing on a restricted neighborhood, thus providing a nuanced degree of outlierness for each data point without relying on a binary classification framework.

The LOF algorithm is predicated on the concept of "local density," where locality is defined by the k-nearest neighbors of a data point. Several core concepts underpin the calculation of LOF:

- The *K-distance* of a point, denoted as K-dist(x_i), represents the distance between the point x_i and its K^{th} nearest neighbor.
- The *K-neighbors* of a point, denoted as K-neighbor(x_i), refers to the set of points whose distance from x_i is less than or equal to the K-distance.
- The *Reachability Distance* between two points x_i and x_j, denoted as $RD(x_i, x_j)$, is defined as the maximum of the K-distance of x_j and the actual distance between x_i and x_j, formally given by:

$$RD(x_i, x_j) = \max(K\text{-dist}(x_j), \text{dist}(x_i, x_j)) \tag{3.1}$$

- The *Local Reachability Density* (LRD) of a point x_i is the inverse of the average reachability distance of x_i from its neighbors. A high average RD implies that the point's neighbors are distant, indicating a low local density around x_i. Conversely, low LRD values suggest that the nearest cluster of points is far from x_i.

The LOF of a point x_i is computed as the ratio between the average LRD of its K neighbors and the LRD of x_i itself. If x_i is not an outlier, its LRD will be similar to that of its neighbors, resulting in a LOF value close to 1. However, if

x_i is an outlier, it will have a significantly lower local density than its neighbors, yielding a LOF value greater than 1.

The choice of the number of neighbors K is a critical parameter that must be carefully tuned within the model. The value of K should be larger than the minimum number of samples required to form a cluster and smaller than the maximum number of nearby samples that could potentially be local outliers. In practice, this information is often unavailable, and selecting K within the range of 10 to 20 generally provides good results. This approach aligns with active learning principles, as the LOF model is updated iteratively with the introduction of new samples into the training set.

3.2.2 QBC: Query by committee

The Query-by-Committee (QBC) method [99] is a prominent active learning technique that leverages an ensemble of models, referred to as committees, to select the most informative data points for labeling. This method hinges on measuring the level of disagreement among the predictions made by the committee members.

In practice, a committee of learners is initially trained on the existing labeled dataset, i.e., the current training set. During each iteration, from a pool of unlabeled data, a single point is selected for labeling based on its potential informativeness. The selection criterion is rooted in the principle of maximal disagreement among the committee members: for each unlabeled point, the committee generates a set of predictions, and the point that elicits the highest disagreement among these predictions is chosen for labeling and subsequently added to the training set.

The measure of disagreement typically employed is the standard deviation of the predictions produced by the committee, particularly in the context of regression tasks. For QBC to be effective, the committee must be composed of heterogeneous learners. This heterogeneity can be achieved in two ways: first, by training a single type of model on different subsets of the labeled data, thus offering varying perspectives of the dataset; second, by incorporating different types of models, each trained on the entire training set. However, it is important to note that models trained on data subsets may be weaker than those trained on the full dataset, and reliance on a single model type could be detrimental, particularly in scenarios where the model performs poorly [52].

The QBC method is an active sampling approach, as it necessitates retraining the committee after each new data point is labeled and incorporated into the training set.

3.2.3 EDD: Euclidean Distance

The Euclidean distance quantifies the proximity of points within the pool of unlabeled samples relative to the existing training set. This distance is calculated using the L2-norm between each unlabeled sample x_i (for $i = 1, \ldots, N$) and the centroid of the training set. The centroid represents the mean position of all the points across all coordinate directions, essentially the center of mass of the dataset.

Mathematically, given a training set Ts with samples $\tilde{x}$ and a pool of unlabeled samples X, the centroid of Ts is defined as

$$\bar{x} = \frac{1}{N} \sum_{\tilde{x} \in Ts} \tilde{x} \tag{3.2}$$

and the Euclidean distance between the centroid and an unlabeled sample x_n is given by

$$EDD(X, x_n) = \|\bar{x} - x_n\|_2 \tag{3.3}$$

The sample requiring labeling is identified as the one with the maximum EDD value. This method constitutes a passive sampling approach, as it does not involve training an underlying model.

3.2.4 GSx, GSy, iGS

The three methods discussed in [151] are here employed here as criteria for the Active Learning setup. "Greedy Sampling on the Input" *(GSx)* is utilized as a Passive Sampling criterion, while "Greedy Sampling on the Output" *(GSy)* and "improved Greedy Sampling on both Input and Output" *(iGS)* are applied as active sampling approaches.

GSx

The *GSx* method determines the distance between the training set and an unlabeled sample based on feature values. Starting from an initial training set, this approach selects a new sample that is the furthest from all previously selected samples in the input space, thereby promoting diversity among the chosen samples. For each unlabeled sample, the minimum distance to each labeled sample is computed, denoted as d_n^x. The sample with the maximum d_n^x is then selected for labeling, making the criterion the maximum of the minimum distances between unlabeled and labeled samples.

If k samples have already been labeled, the distance between each of the remaining $N - k$ unlabeled samples x_n (where $n = k + 1, \ldots, N$) and each of the k labeled samples is computed as:

$$d_{nm}^x = \|x_n - x_m\|, \; m = 1, \ldots, k; \; n = k + 1, \ldots, N \tag{3.4}$$

Then, d_n^x, the minimum distance from x_n to the k labeled samples, is given by:

$$d_n^x = \min_m d_{nm}^x,\ n = k+1, \ldots, N \tag{3.5}$$

The sample with the maximum d_n^x is selected for labeling.

GSy

The *GSy* approach measures diversity in the output space. For each iteration, a new sample is chosen based on its distance from all previously selected samples in the output space. This method requires constructing a regression model $f(x)$, which is a parameter to be optimized. For each unlabeled sample x_i, the minimum distance between the model's prediction $f(x_i)$ and each label of the training samples is computed, denoted as d_n^y. The sample with the maximum d_n^y is selected, making the criterion the maximum of the minimum distances between the prediction and the actual labels.

Assuming that k samples with outputs y_m (where $m = 1, \ldots, k$) have been labeled, and a regression model $f(x)$ is constructed, the distance between each of the remaining $N - k$ unlabeled samples x_n (where $n = k+1, \ldots, N$) and each of the k outputs is calculated as:

$$d_{nm}^y = \|f(x_n) - y_m\|,\ m = 1, \ldots, k;\ n = k+1, \ldots, N \tag{3.6}$$

Then, d_n^y, the minimum distance from $f(x_n)$ to the k outputs, is:

$$d_n^y = \min_m d_{nm}^y,\ n = k+1, \ldots, N \tag{3.7}$$

The sample with the maximum d_n^y is selected for labeling.

iGS

The *iGS* method evaluates distances between training and unlabeled samples in both the input and output spaces, integrating elements of GSx and GSy. For each unlabeled sample, distances are computed in both spaces, and the score for each sample is determined by the minimum of the product of distances in the input and output spaces.

Assuming k samples have already been labeled with labels y_n (where $n = 1, \ldots, k$), the distances d_{nm}^x and d_{nm}^y are computed for each of the remaining $N-k$ unlabeled samples x_n (where $n = k+1, \ldots, N$). The combined score d_n^{xy} is given by:

$$d_n^{xy} = \min_m \left(d_{nm}^x \cdot d_{nm}^y\right),\ n = k+1, \ldots, N \tag{3.8}$$

The sample with the maximum d_n^{xy} is selected for labeling.

3.2.5 Hausdorff Distance

The Hausdorff distance *HD* [74, 91, 83] quantifies the extent to which two subsets of a metric space are separated from each other. Two sets are considered close under the HD if every point in either set is near some point in the other set. More formally, it represents the maximum of all distances from a point in one set to the nearest point in the other set. This distance measure has been applied in computer vision to assess image similarity [74], evaluate generative models [91], and in face recognition [83].

The directed HD can be employed as a standard distance metric to compare the training set with the test set. For two finite point sets $A = \{a_1, \ldots, a_p\}$ and $B = \{b_1, \ldots, b_q\}$, the HD is defined as:

$$H(A, B) = \max(h(A, B), h(B, A)) \tag{3.9}$$

where

$$h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\|_2 \tag{3.10}$$

and $\|\cdot\|_2$ denotes the L2 or Euclidean norm. The term $h(A, B)$ is referred to as the *directed* Hausdorff distance from A to B. It measures the distance from the point $a \in A$ that is furthest from any point in B to its closest neighbor in B, according to the given norm.

3.3 Multi Task Learning

Multi-Target or Multi-Output Regression pertains to regression tasks where multiple real numerical values are predicted for each input data. Given an input space $D \in \mathbb{R}^n$ and an example $x = (x_1, x_2, \ldots, x_n) \in D$, the goal is to predict a target vector $y = (y_1, y_2, \ldots, y_m)$ in the output space $Y \in \mathbb{R}^m$. Thus, a multi-target model can be viewed as a function mapping $\mathbb{R}^n$ to $\mathbb{R}^m$.

Several approaches can be employed to tackle multi-target regression problems. One common method involves transforming the multi-output problem into multiple single-target problems. The simplest approach treats each target variable independently, training m separate models, where each model $i \in \{1, \ldots, m\}$ uses the input tuple (x, y_i). However, in many applications, targets are not independent; thus, separate models fail to leverage the potential correlations among targets, resulting in a loss of valuable information in the output space. This method, known as the "Single Target method" [34], involves converting the multi-output problem into independent single-output problems and then concatenating the individual predictions.

Two advanced approaches are discussed in [129]: the Regressor Chain (RC) and Multi-Target Regressor Stacking (MT-RS). The Regressor Chain is a problem-transformation technique that sequentially links single-target models, forming a chain of regressors $h = (h_1, h_2, \ldots, h_m)$. Each model in the chain utilizes the predictions from its predecessors.

The specific Regressor Chain used in this work is the Corrected Regressor Chain, as outlined in [129].

The Corrected Regression Chain method enhances the traditional regression chain approach. It was introduced to address the limitations of the basic regression chain model, particularly its sensitivity to the order of predictions.

During training, the Regressor Chain relies on the true target values from the training set to fit the chained models. However, at prediction time, models use estimated values, which deviates from the independent and identically distributed assumption due to the differing distributions of true and estimated values. This discrepancy may affect prediction performance. The Corrected Regressor Chain addresses this by using predictions from earlier models in the chain rather than true values, employing an internal f-fold cross-validation approach, specifically 5-fold cross-validation, to generate the input space for the subsequent models. This adjustment helps mitigate the bias introduced by possible distribution shift and mitigate the order of predictions, leading to more robust predictions.

Given a predefined order C for predicting the labels (assuming $C = (y_1, y_2, \ldots, y_m)$), the initial model in the chain is trained on $D_1 = (X, Y_1)$. Subsequent models are trained on augmented datasets incorporating the predicted labels from prior steps: h_2 is trained on $D_2 = (X, \hat{Y}_1)$, h_3 is trained on $D_3 = (X, \hat{Y}_1, \hat{Y}_2)$, and so forth. This chaining process allows models to exploit relationships within the output space.

The Multi-Target Regressor Stacking (MT-RS) approach is a two-stage method. In the first stage, m "weak" regressors are trained independently, one per label, using a Single-Target approach, resulting in a list $h = (h_1, h_2, \ldots, h_m)$, where each regressor h_i is trained on the dataset (X, y_i). These models produce an augmented dataset $X^{aug} = (X, \hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_m)$, where $\hat{Y}_i$ represents the predicted label from regressor h_i.

In the second stage, X^{aug} serves as the input for m meta-models, forming another list $h^* = (h_1^*, h_2^*, \ldots, h_m^*)$, where each h_i^* is trained on (X^{aug}, Y_i) to predict target y_i. The MT-RS method leverages the idea that a second-stage model can correct the predictions of the first-stage models by incorporating information about the predictions from the other first-stage models [129]. A detailed description of the algorithms for RC and MT-RS can be found in [130].

3.4 Dataset Shift and generalization problem

Dataset shift, also termed concept shift or drift, is a common challenge in predictive modeling, where the joint distribution of input and output variables diverges between the training and testing stages. This issue can emerge due to a variety of factors, including data sparsity, biases in the experimental setup, or inherent temporal changes in the problem domain. In real-world applications, the assumption of independent and identically distributed (i.i.d.) data between training and test sets often fails, leading to degraded performance of ML models on specific subsets of the data. Dataset shift can manifest in both supervised and semi-supervised learning contexts, typically impacting the labels in the task. The primary types of dataset shift are:

- Covariate Shift
- Prior Probability Shift
- Concept Drift
- Internal Covariate Shift (a variant of Covariate Shift)

Covariate Shift arises when the distribution of input features differs between the training and testing phases, usually due to changes in latent variables that may be temporal or spatial in nature, creating a non-stationary environment. Formally, covariate shift is expressed as:

$$\mathcal{D}(x)_{train} \neq \mathcal{D}(x)_{test} \tag{3.11}$$

Prior Probability Shift occurs when the distribution of the dependent variable, such as the labels or class distributions $\mathcal{D}(y)$, differs between training and testing datasets:

$$\mathcal{D}(x|y)_{train} = \mathcal{D}(x|y)_{test}, \quad \mathcal{D}(y)_{train} \neq \mathcal{D}(y)_{test} \tag{3.12}$$

This type of shift can be viewed as a class imbalance in the dataset.

Concept Drift refers to alterations in the statistical characteristics of the target variable over time, leading to a decline in prediction accuracy as time progresses. Concept drift is characterized by:

$$\mathcal{D}(y|x)_{train} \neq \mathcal{D}(y|x)_{test}, \quad \mathcal{D}(x)_{train} = \mathcal{D}(x)_{test} \tag{3.13}$$

Internal Covariate Shift is specifically associated with Deep Neural Networks, where changes in the distribution of activations from one hidden layer can influence subsequent layers, potentially compromising training efficiency.

The main drivers of dataset shift include sample selection bias and non-stationary environments. Sample selection bias can emerge in cross-validation frameworks, often going unnoticed [101]. The use of k-fold cross-validation for model performance estimation is common, where model selection often involves averaging performance metrics across multiple cross-validation runs (e.g., 2x5, 5x2, or 10x2 folds), assuming that the partitions are independently constructed.

Non-stationary environments, on the other hand, occur when the input data distribution changes due to temporal or spatial factors [110]. Traditional machine learning assumes a static input data distribution over time. However, when the distribution shifts during testing or deployment, a covariate shift is observed. Addressing this challenge requires computationally efficient methods capable of detecting shifts in underlying data distributions. Real-time or online classification tasks often encounter dataset shifts, prompting various strategies for mitigation, such as offline retraining of classifiers with updated data or incremental online updates of the model [7].

Active learning techniques provide an effective strategy to address dataset shift by constructing an optimal training set that generalizes well to new, unseen data [137]. Additionally, Novelty Detection methods can be employed to model and adapt to the changing distribution of data sources [153].

3.5 Dimensionality Reduction

Dimensionality reduction plays a pivotal role in machine learning (ML) by decreasing the number of features that a model needs to process. Its primary objectives include enhancing model performance, diminishing computational complexity, and alleviating the risk of overfitting.

This reduction in dimensionality is typically achieved through two principal approaches: Feature Selection (FS) and Feature Extraction (FE) [69]. FS focuses on identifying and selecting a subset of the most pertinent features from a larger set of potential variables, with the goal of eliminating redundant or irrelevant features that may contribute to overfitting or reduced generalizability [67]. Conversely, FE aims to reduce the dataset's dimensionality by deriving new features from the existing ones. Techniques such as Principal Component Analysis (PCA) are commonly employed in FE to achieve a more compact representation of the data [77].

FS methods can be categorized into three main types: Filter Methods, Wrapper Methods, and Embedded Methods [67].

Filter methods evaluate feature relevance using statistical measures, ranking features based on their relationships with the target variable. For instance, correlation-based FS techniques assess the statistical associations between each

feature and the target variable. Common metrics include the Pearson correlation coefficient and Spearman Rank [121], with features exhibiting the highest correlations being selected [76] in a univariate supervised context. Alternatively, features with high inter-correlation can be filtered out in an unsupervised manner.

Wrapper Methods utilize an underlying ML model to assess various subsets of features, selecting the subset that yields the best performance as determined by error metrics on a designated test set, often through cross-validation. While computationally more intensive than filter methods, wrapper methods often produce superior results. Recursive Feature Elimination (RFE) is a notable example of such an approach [68, 76]. RFE involves iteratively training an ML model on the complete feature set, ranking features according to their importance, removing the least important features, and repeating this process until a predetermined stopping criterion is met, such as a specified number of features or a threshold in prediction error. Feature importance can be assessed using metrics like the magnitude of weights in linear regression models or the reduction in impurity (e.g., Gini impurity or entropy) in Decision Trees and Random Forests [**Breiman2001**, 112].

Embedded Methods integrate feature selection directly into the model training process, thus eliminating the need for a separate feature selection step. Examples include Decision Trees and Random Forests [**Breiman2001**], as well as linear models incorporating regularization techniques.

PCA is a classic FE technique aimed at reducing dimensionality by identifying a low-dimensional representation that retains most of the original data's variance. Originating from Karl Pearson's work in the early 20th century [77], PCA is particularly effective in simplifying models by reducing highly correlated features and aiding in data visualization through projection into lower-dimensional spaces.

The PCA procedure involves the following steps for a dataset with N samples:

- Standardize the data by subtracting the mean and dividing by the variance of each feature to ensure comparability across features.
- Compute the covariance matrix:

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(X_i - \bar{X})(X_i - \bar{X})^T, \tag{3.14}$$

  where $\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$ denotes the mean of the dataset.
- Determine the eigenvectors and corresponding eigenvalues of the covariance matrix.
- Rank the eigenvalues in descending order and select the top k eigenvectors associated with the largest eigenvalues as the principal components.

- Construct the transformation matrix W with columns representing the selected k eigenvectors. The reduced-dimensional representation of the data is then computed as:

$$X_t = XW. \tag{3.15}$$

PCA has proven valuable in various contexts, including its effective application in studies related to SMONs [48].

Applying dimensionality reduction techniques, such as PCA, can significantly benefit even complex models like Deep Learning models by reducing computational and memory requirements while improving interpretability. Additionally, these techniques can help mitigate the adverse effects of noisy or irrelevant features on prediction performance.

3.5.1 Compressed Sensing

Signal processing has traditionally focused on the acquisition, analysis, and reconstruction of signals using techniques like Fourier analysis and sampling theory. Conventional methods typically assume that signals are uniformly and densely sampled, resulting in the Nyquist-Shannon sampling criterion. However, in numerous practical scenarios, the acquisition and storage of vast amounts of data become both impractical and costly [95]. Compressive Sensing (CS) [43] addresses this issue by establishing that signals with sparse representations can be accurately reconstructed from a reduced number of linear measurements. In high-dimensional systems, significant behavior can often be captured using sparse models, where the number of parameters is far fewer than the state variables [61].

Specifically, CS outlines the conditions necessary to recover a sparse vector $x \in \mathbb{R}^n$, containing $k < n$ non-zero components, from a linear regression equation $y = Ax + \eta \in \mathbb{R}^m$, where $m < n$ and η represents measurement noise [**fou13**]. The development of efficient algorithms, such as Basis Pursuit and Orthogonal Matching Pursuit, has facilitated the practical application of CS [136]. Additionally, regularized linear models, as discussed in the following section, are effective in promoting sparsity, with the LASSO (Least Absolute Shrinkage and Selection Operator) [134] being particularly notable.

Over time, CS has been applied across various domains, including medical imaging, radar systems, and communications [155, 95].

3.6 Machine Learning model used

Various machine learning models will be employed to address the regression task involved in predicting the actual MCU $F_{\max}$. The most prominent and relevant models in this context are outlined and discussed.

3.6.1 Linear Models

Linear models generate outputs through a linear combination of input features, each weighted by a set of coefficients. The training process involves determining these weights by solving an optimization problem. The fundamental linear regression model employs Least Squares estimation, which seeks to minimize the residual sum of squares between the actual target values in the training set and the values predicted by the linear model. Mathematically it solve a minimization problem of the form:

$$\min_{w} ||Xw - y||_2^2 \tag{3.16}$$

In which $w = (w_1, \ldots, w_p)$ are the coefficient of the model, X are the features and y the (true) labels. This is called ordinary least-square (OLS) linear regression objective function. If the model is affected by overfitting, regularization can be applied.

Regularized Linear Models

Regularized linear models are a class of linear regression models that incorporate regularization techniques to improve accuracy and prevent overfitting. These techniques add a OLS objective function, influencing the optimization process to reduce the number of active parameters. The optimization problem is

$$\min_{w} \frac{1}{2n} \|Xw - y\|_2^2 + \alpha \sum_{j=1}^{m} \text{pen}(|w_j|) \tag{3.17}$$

where the first component is the OLS objective and the second is a penalty term on the coefficient of the model. Here, $X \in R^{n,m}$ is the observations' matrix, $y \in R^n$ is the regression target, $w \in R^m$ is the coefficients vector, and α is the regularization strength hyper-parameter. The penalty is usually an ℓ_p norm of the coefficient vector w, defined as

$$\|w\|_p = \left(\sum_{i=1}^{m} |w_i|^p \right)^{\frac{1}{p}}. \tag{3.18}$$

The convex ℓ_1 and ℓ_2 norms are popular regularization techniques, and they give rise to ridge regression [73] and LASSO [134], respectively. While ℓ_2 regularization has mainly a smoothing effect, ℓ_1 regularization has a sparsity promoting effect, that yields an effective variable selection in linear regression. More recently, non-convex regularization penalties [64] have been introduced, which improve the variable selection performance with respect to classic ℓ_1 regularization. Examples of effective non-convex regularization penalties are Log Sum Penalty (LGP, [106]), Minimax Concave Penalty (MCP, [90]), Smoothly Clipped Absolute Deviation (SCAD, [154]) and the ℓ_p norm with $0 < p < 1$ [64].

Convex Regularization

The LASSO problem is (3.17) with ℓ_1 regularization, i.e., $\text{pen}(|w_j|) = |w_j|$. The ℓ_1 regularization encourages sparsity in the model, leading several coefficients to become exactly zero. LASSO is a convex problem, therefore it is mathematically affordable. It represents an effective recovery strategy for CS, i.e., for linear regression problems with $w \in \mathbb{R}^m$, $y \in \mathbb{R}^n$ and $n < m$. LASSO is closely related to the principles of compressed sensing, particularly in the context of sparse signal recovery: promoting sparsity in the regression coefficients, LASSO effectively selects a subset of features contributing most to the prediction.

In ridge regression [73], the optimization problem is (3.17) with $\text{pen}(|w_j|) = w_j^2$. The ℓ_2 regularization helps in reducing the impact of multicollinearity in the data, penalizing large coefficients and shrinking them towards zero, but it does not lead to sparsity in the model.

ElasticNet [161] is a linear regression model trained with both ℓ_1 and ℓ_2 norm regularization of the coefficients. This combination allows for both learning a sparse model (like in LASSO) and maintaining the regularization properties of ridge regression. The corresponding optimization problem is

$$\min_w \frac{1}{2n} \|Xw - y\|_2^2 + \alpha\rho \|w\|_1 + \frac{\alpha(1-\rho)}{2} \|w\|_2^2.$$

The convex combination of ℓ_1 and ℓ_2 is controlled using α and ρ parameters.

Orthogonal Matching Pursuit

Beyond convex regularization, greedy algorithms are traditionally used for CS. In particular, OMP [114] approximates the fit of a linear model with constraints imposed on the number of non-zero coefficients (i.e., the ℓ_0 pseudo-norm), finding a coefficients vector with a fixed number of non-zero elements:

$$\arg\min_w \|y - Xw\|_2^2 \text{ subject to } \|w\|_0 \leq n_{\text{nonzero_coefs}}. \tag{3.19}$$

OMP is based on a greedy algorithm that includes at each step the features most highly correlated with the current residual, i.e. the difference between an observed value y_i and its corresponding predicted value $\hat{y}$. Once the feature is selected, the signal is orthogonally projected to the span of the selected feature, the residual is recomputed, and the process repeats. OMP can be considered as a convex approach because each step in the algorithm involves solving a convex least squares problem. However, the overall optimization problem solved by OMP is non-convex.

Non-Convex Regularization

In the last years, continuous non-convex sparsity-promoting regularization has been attracting substantial attention [64]. While solving LASSO is affordable thanks to its convexity, the global minimum suffers from bias with respect to the true solution, due to the penalty term. In contrast, non-convex regularization, with shape closer to the ℓ_0 norm, may reduce the bias [90]. Due to the non-convex nature of the objective function, no theoretical assurances are provided regarding the attainment of the global minimum [64]. However, research has demonstrated that non-convex linear models exhibit superior performance within a fixed time frame, achieving higher accuracy levels in less time compared to convex optimization methods [33, 64]. Moreover, it can produce sparser models, which is desirable for interpretability and feature selection, and it may accelerate the rate of convergence of the solution algorithms [45]. The use of non-convex regularization has become popular in CS, in the framework of ℓ_1 reweighting techniques [60], while recently it has been also analyzed for pruning purposes in deep neural networks [62].

Among the most popular non-convex regularizers, MCP approximates the ℓ_1 penalty when α is arbitrarily large. In particular, MCP maintains the shape of the ℓ_1 norm around the origin while approximating the scaled ℓ_0 norm away from the origin [90]. The MCP penalty is

$$\text{pen}(w) = \begin{cases} \alpha w - \frac{w^2}{2\gamma} & \text{if} \quad w \leq \alpha\gamma \\ \gamma\frac{\alpha^2}{2} & \text{if} \quad w > \alpha\gamma \end{cases} \tag{3.20}$$

in which $\alpha > 0$ is the regularization strength and $\gamma > 0$ is an hyper-parameter.

SCAD [154] penalizes large coefficients less severely than LASSO, leading to more stable and accurate model estimation, especially when dealing with correlated predictors. The SCAD penalty is

$$\text{pen}(w) = \begin{cases} \alpha w & \text{if} \quad w \leq \alpha \\ 2\alpha\gamma w - w^2 - \frac{\alpha^2}{2}(\gamma - 1) & \text{if} \quad \alpha < w \leq \alpha\gamma \\ \alpha^2\frac{\gamma+1}{2} & \text{if} \quad \alpha\gamma < w \end{cases} \tag{3.21}$$

where $\alpha > 0$, $\gamma > 0$.

LGP is often adopted as a replacement for the ℓ_0 pseudo-norm in CS and low-rank optimization. [106] The LGP penalty is

$$\text{pen}(w) = \sum_{j=1}^{m} \log\left(1 + \frac{|w_j|}{\epsilon}\right) \tag{3.22}$$

where $\alpha > 0$, $\gamma > 0$.

Among the ℓ_p regularizers, with $0 < p < 1$, the case $p = \frac{1}{2}$ is often used [**fou13**, 64].

3.6.2 SVM

Support Vector Machines (SVMs) [70, 126, 135] represent a class of supervised machine learning models applied across various domains, including classification, regression, outlier detection [153], and pairwise ranking. Originally developed as binary classification models, SVMs aim to separate classes by identifying the hyperplane that maximizes the margin, defined as the minimum distance between any data point and the hyperplane. The primary objective is to maximize the separation width between the classes. The optimization problem associated with this task is a Quadratic Program (QP), characterized by a quadratic objective function and linear constraints. For the classification scenario (Support Vector Classifier, or SVC), the problem is mathematically formulated as:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i \\ \text{s.t.} \quad & y_i[< w, x_i > +b] \geq 1 - \xi_i, \\ & \xi \geq 0 \end{aligned} \tag{3.23}$$

Here, w and b denote the model parameters, while ξ is a slack variable that allows for the misclassification of some points, accommodating cases where the dataset is not linearly separable.

SVMs can be extended to handle non-linear classification tasks through the application of the *Kernel Trick*, section 3.6.4. This approach involves implicitly transforming the input features by mapping the original feature space into a higher-dimensional space, where the problem becomes (ideally) linearly separable.

Support Vector Regression (SVR) [11, 58] is a powerful regression technique derived from SVM. SVR extends the principles of SVM to regression problems by seeking to find a function that approximates the target data while maintaining a specified margin of tolerance for errors. The primary objective of SVR is to minimize the prediction error while maintaining model complexity, which is achieved through a cost function that incorporates both the error of the predictions and a regularization term. Similarly to SVM, SVR is particularly adept at handling non-linear relationships through the use of kernel functions. This flexibility, combined with its robustness to outliers, makes SVR a popular choice in various fields, including finance, engineering, and bioinformatics.

For a comprehensive mathematical treatment of Support Vector Machines (SVMs), see [126].

3.6.3 Tree Based Estimator

Tree-based machine learning models are widely recognized and frequently employed for both classification and regression tasks. These models operate by applying a sequence of simple decision rules, typically in the form of if-else statements, derived from the training dataset. The construction of such models is recursive, involving the iterative splitting of the training examples based on the values of one feature at a time at each node. These decision rules can be visually represented as the nodes of a tree, which gives rise to the term "*decision tree methods.*" Beginning at the root node, an instance is predicted by traversing the tree, moving through decision nodes that pose yes-or-no questions, until a final prediction is reached. Decision trees [118] are versatile, functioning effectively in both classification and regression contexts.

The advantages of tree-based models are notable and include:

- Interpretability: The model's decision-making process is transparent, allowing for straightforward understanding of how a particular prediction or decision is derived.
- Feature Importance: Decision trees can identify and prioritize the most significant features, often using them as the root node.
- Speed: These algorithms are efficient in making predictions.
- Minimal Preprocessing: Tree-based models require minimal data preprocessing, such as normalization.

Constructing a decision tree involves selecting which features to split on and determining when to stop splitting. When choosing features for splitting, the objective is to select the feature that generates the most homogeneous subsets of data. The splitting criterion is based on selecting the feature and the corresponding value that produces the most homogeneous branches, thereby aiming to minimize a cost function (such as the Gini Index [149] or Entropy [97] in classification, and Mean Squared Error (MSE) or Mean Absolute Error (MAE) in regression tasks [53]).

A common approach involves minimizing entropy—a measure of disorder within the dataset—while maximizing information gain, which reflects the reduction in entropy achieved by splitting on a particular feature [108].

Decision trees can also serve as weak learners in ensemble methods, notably in the construction of Random Forests. A Random Forest [**Breiman2001**] is a machine learning model composed of multiple decision trees, each built from a subset of the training data drawn with replacement. The best split at each node

can be determined either from all features or from a randomly selected subset of features of a given size. Final predictions are made by aggregating the predictions from all trees—using majority voting in classification tasks or averaging in regression problems. The inclusion of a large number of trees (ranging from dozens to hundreds) enables Random Forests to capture non-linear relationships, enhancing their predictive power and accuracy.

3.6.4 Kernel Ridge Regression

Kernel Ridge Regression (KRR) [127, 152] is an extension of Ridge Regression that leverages the kernel trick to model non-linear relationships within data. The KRR model can be expressed as:

$$\hat{y} = K(\alpha + I\lambda) \tag{3.24}$$

where K is the kernel matrix, α is the vector of dual coefficients, and λ is the regularization parameter. The kernel function $K(x_i, x_j)$ allows the algorithm to operate in a high-dimensional feature space without explicitly computing the coordinates of the data in that space, making it computationally efficient for non-linear problems [127]. Commonly used kernels include the linear, polynomial, and radial basis function (RBF) kernels. KRR is particularly useful in scenarios where the relationship between the features and the target variable is highly non-linear [103].

3.6.5 Multivariate Adaptive Regression Splines (MARS)

Multivariate Adaptive Regression Splines (MARS) [63] is a non-parametric regression technique that models relationships by fitting piecewise linear regressions. It automates the selection of interaction terms and the detection of non-linearities in data. MARS constructs models of the form:

$$\hat{y} = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(x) \tag{3.25}$$

where β_0 is the intercept, β_m are the coefficients, and $h_m(x)$ are the basis functions. These basis functions are typically hinge functions of the form $\max(0, x-t)$ or $\max(0, t-x)$, where t is a knot location determined during model training. The MARS algorithm proceeds in two phases: the forward phase, which adds basis functions to the model, and the backward phase, which prunes the least effective basis functions to avoid overfitting [63]. MARS is especially advantageous for high-dimensional data due to its ability to capture complex interactions between variables without manual specification [69].

3.6.6 Neural Networks

Neural networks, frequently referred to as artificial neural networks (ANNs) [103, 87, 29], constitute a fundamental class of algorithms within the field of machine learning, serving as the cornerstone for deep learning. Their architecture, inspired by the structural and functional aspects of the human brain, comprises interconnected units known as neurons (or nodes) and the connections between them, termed synapses. Neural networks excel in pattern recognition tasks and have become integral to a wide array of machine learning applications, spanning supervised, unsupervised, semi-supervised, and reinforcement learning.

The architecture of a neural network is organized into layers, with the outermost layers designated as the input and output layers, and the intermediate layers referred to as hidden layers. Each neuron within these layers is connected to others through weighted connections, where the weights represent the strength of the influence between neurons. The computational function of a neuron involves the calculation of a weighted sum of its inputs, followed by the application of an activation function. This activation function, typically non-linear, enables the network to model complex, non-linear relationships within the data. The mathematical formulation of the output of a neuron is expressed as:

$$\text{output} = f_{\text{activation}}\left(\sum_{i=1}^{n} w_i \cdot \text{input}_i + \text{bias}\right) \tag{3.26}$$

where w_i are the weights associated with each input input_i, and the bias term allows for an affine transformation.

When multiple layers are stacked, the resulting architecture is termed a deep neural network (DNN). A notable instance of such an architecture is the multilayer perceptron (MLP), which is a fully connected network comprising several layers. The universal approximation theorem [88] asserts that, with a sufficient number of layers and neurons per layer, a neural network can approximate any arbitrary function, demonstrating its powerful modeling capabilities. Figure 3.1 illustrates an example of a fully connected neural network, which, due to the extensive number of trainable parameters, requires a substantial amount of data for effective training.

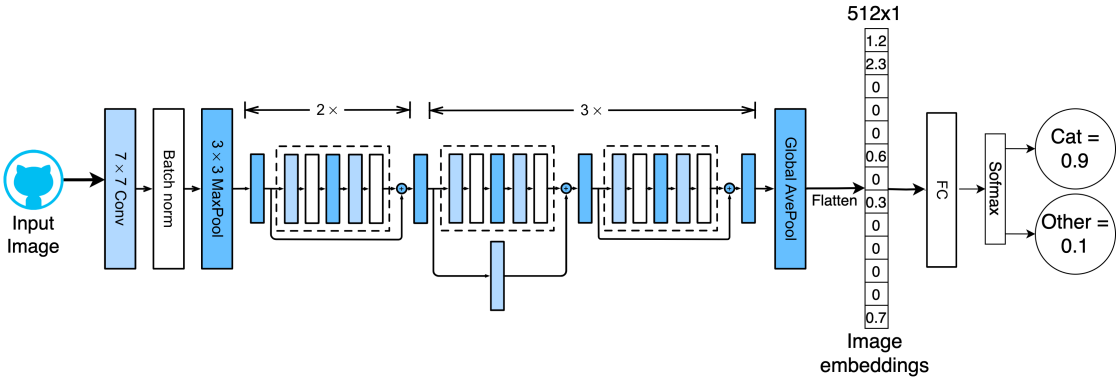


Figure 3.1: Example of a ResNet18 neural network applied to a binary classification problem with image inputs. The network outputs the probabilities associated with each class.

A common choice for the activation function in neural networks is the Rectified Linear Unit (ReLU) [1], known for its simplicity and effectiveness, as summarized in Table 3.1. In classification tasks, the final layer typically employs the Softmax function to convert the network's output into a probability distribution over the predicted classes. Another activation function, the hyperbolic tangent (TanH), which outputs values in the range of -1 to 1, has fallen out of favor due to its propensity to cause the vanishing gradient problem [71, 32], thereby complicating the training process.

ReLU	$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}, \quad i = 1, ..., J$
tanh	$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

Table 3.1: Examples of activation functions commonly used in neural networks.

The process of training a neural network is framed as an optimization problem, where the objective is to minimize a loss function over the training data. The choice of the loss function varies with the task: for regression tasks, the mean squared error (MSE) loss [46, 53] is prevalent, while cross-entropy loss [97] is favored for classification tasks . Table 3.2 provides the mathematical formulations of these loss functions.

Convolutional Neural Networks (CNNs) [133] have emerged as one of the most

MSE / L2 Loss / Quadratic Loss	$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
Binary Cross Entropy (average reduction on higher dimensions)	$\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C}\hat{y}_i \log(y_{i,j})$
Categorical Cross Entropy (sum reduction on higher dimensions)	$-\sum_{i=1}^{N}\hat{y}_i + \log\left(\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\right)$

Table 3.2: Examples of loss functions utilized in neural network training. Here, y represents the predicted output of the network, while $\hat{y}$ denotes the true target values, and C is the number of classes.

influential architectures within deep learning, particularly excelling in the domain of computer vision. Their architecture, composed of convolutional, pooling, and fully connected layers, enables CNNs to achieve state-of-the-art performance across a variety of tasks, including image classification, object detection, and semantic segmentation [80, 4]. The convolutional layers, equipped with learnable filters, are designed to automatically extract hierarchical feature representations from raw input data, such as images. Following the convolutional operations, activation functions like ReLU introduce non-linearities, allowing the network to model complex patterns.

Subsequent to convolutional layers, pooling layers perform downsampling operations on the feature maps, reducing their spatial dimensions while maintaining essential information. This downsampling enhances the network's translation invariance and computational efficiency. The final stage of the CNN consists of fully connected layers that interpret the high-level features for classification or regression tasks, depending on the application. These fully connected layers ensure that every neuron in one layer is linked to every neuron in the subsequent layer, facilitating the final decision-making process of the network.

In addition to their success in supervised learning tasks, CNNs are also adept at unsupervised learning. Convolutional autoencoders, for instance, are employed to learn compact, meaningful representations of unlabeled data. These autoencoders consist of an encoder network that compresses the input data into a lower-dimensional representation, and a decoder network that reconstructs the original input from this compressed form. The ability of CNNs to capture the most salient features of the data makes them valuable in tasks such as dimensionality reduction, feature extraction, and image denoising. Moreover, these unsupervised CNNs often serve as pre-training mechanisms for supervised CNNs, particularly in scenarios where labeled data is scarce, thereby improving generalization and overall performance.

CNNs have primarily been utilized in image analysis due to their capability to

capture spatial structures in data [13, 4]. The convolutional kernels in CNNs are designed to exploit two critical properties of images: local connectivity and spatial locality. Local connectivity refers to the fact that each kernel operates over a small region of the input image, preserving spatial relationships. Spatial locality, on the other hand, assumes that pixels in close proximity are more likely to be related and should be processed together to yield meaningful feature representations. Through this approach, CNNs effectively detect features such as edges, textures, and shapes, which are vital for understanding the visual content.

Recent advancements have extended the application of CNNs beyond image data to tabular data. Research has demonstrated that tabular data can be transformed into image-like structures, enabling the application of CNNs to non-visual tasks [35, 29, 13]. Alternatively, one-dimensional convolutions can be applied directly to the columns of a dataset, treating each instance as a one-dimensional image. This flexibility in adapting CNNs to various data formats underscores their potential in capturing complex, non-linear relationships across diverse domains.

3.6.7 Transfer and Semi-Supervised Learning

Transfer learning has established itself as a crucial methodology in ML, enabling enhanced task performance through the utilization of knowledge acquired from distinct but related problems [92, 160]. This methodology is particularly effective within the context of Deep Neural Networks, where pre-trained models—originally trained on vast datasets with millions of examples—serve as the foundation for new tasks [92, 160].

Rather than initiating the training process from scratch, these pre-trained models are fine-tuned for new applications. During this fine-tuning phase, the early layers of the pre-trained network are typically kept fixed, preserving the integrity of the learned features. These frozen layers function as feature extractors, identifying general patterns that are applicable across a range of datasets and tasks [66]. The adaptability of the network is further refined by restricting training to the final linear layer, which maps the extracted features to task-specific labels, thereby facilitating a seamless transition between generalized knowledge and task-specific requirements.

In tandem with transfer learning, SSL presents a unique paradigm within ML [49, 2]. Operating in the intermediate space between unsupervised learning, which lacks labeled data, and supervised learning, which depends entirely on labeled examples, SSL becomes particularly valuable when labeling data is expensive or impractical, yet large amounts of unlabeled data are available. The core objective of SSL is to leverage both labeled and unlabeled data to train ML models, thereby enhancing data utilization and reducing dependency on labeled data. This

approach can be seen as a form of transfer learning, where the transfer occurs between unlabeled and labeled data.

A variety of SSL techniques have been developed to address these challenges [49, 156, 159]. A basic technique, "*Pseudo-Labeling*," involves training a model on a small labeled dataset, then applying the trained model to predict labels for the unlabeled data, which are subsequently used to refine the model. This iterative process includes both the original labeled data and the newly pseudo-labeled data in subsequent training cycles.

A more advanced method, "*Self-Training*," iteratively applies Pseudo-Labeling but selectively includes only the most confident pseudo-labeled examples at each iteration [49]. This selective inclusion enhances the robustness of the final model by incrementally expanding the labeled dataset with highly reliable pseudo-labels.

The effectiveness of these SSL approaches hinges on the model's ability to estimate confidence intervals or probabilities of correctness for its predictions, rendering them particularly suitable for classification tasks [159]. Although less common, there are explorations into SSL applications for regression tasks as well [84].

The interplay between Transfer Learning and SSL significantly bolsters the learning capabilities of ML models. Transfer Learning exploits pre-trained models to improve performance across new domains, while SSL facilitates efficient knowledge extraction by navigating between labeled and unlabeled data. Together, these methodologies contribute to the advancement of ML by reducing the necessity for large labeled datasets, expediting training processes, and enhancing generalization across tasks. The synergistic application of these techniques, particularly when combined with the adaptability of convolutional neural networks, underscores the extensive potential within the field, enabling improvements in training efficiency, data utilization, and model performance.

3.7 Model Evaluation

After a machine learning model is trained, the quality of the model as to be evaluated on a test set of n samples, with true labels Y and predicted labels $\hat{Y}$. The use of a test set, separated from the training set, is needed in order to provide an unbiased estimation of the ability of the model to well generalize on new data, unseen at training time. Different metrics exist to measure the quality of the model, depending on the task we are facing (regression or classification).

3.7.1 Regression

The main metrics in the regression context are the *Mean Squared Error MSE*, *Mean Absolute Error MAE*, *normalized root mean squared error nRMSE*, the *normalized root mean absoluted error nRMAE* [53, 46].

MAE, MSE, nRMSE and nRMAE

The *MSE* is computed as:

$$MSE(Y - \hat{Y}) = \frac{1}{n}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2 \tag{3.27}$$

while the *MAE* is computed as:

$$MAE(Y - \hat{Y}) = \frac{1}{n}\sum_{i=1}^{N}(Y_i - \hat{Y}_i) \tag{3.28}$$

The *nRMSE* and *nRMAE* are the squared root of MSE and MAE, normalized on the mean of the label Y, that in our case is the mean of operating frequency. The normalization permits to express these performances measure on a percentage scale. nRMSE and nRMAE are computed as

$$nRMSE(Y - \hat{Y}) = \frac{\sqrt{MSE(Y - \hat{Y})}}{mean(Y} \tag{3.29}$$

$$nRMAE(Y - \hat{Y}) = \frac{\sqrt{MAE(Y - \hat{Y}}}{mean(Y)} \tag{3.30}$$

$$\tag{3.31}$$

3.7.2 R^2

The R^2 [53] score is a metric that evaluates the proportion of variance in the dependent variable, y, that is accounted for by the independent variables, x, within a model. This metric functions as an indicator of the model's goodness of fit, reflecting the model's ability to accurately predict new, unseen data by measuring the proportion of variance that is explained by the model.

The R^2 score is defined mathematically as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3.32}$$

where y_i represents the actual values, $\hat{y}_i$ denotes the predicted values, and $\bar{y}$ is the mean of the actual values.

The value of R^2 ranges between 0 and 1. A score of 1 indicates a perfect fit, meaning the model explains 100% of the variance in the dependent variable. Conversely, a score of 0 suggests that the model does not explain any of the variance, equivalent to the performance of a constant model that always predicts the mean of y. Thus, R^2 provides an essential measure of the model's predictive accuracy in relation to the variability inherent in the data.

Guardband

The guardband G [150] is defined as how much increase the amount of minimum product specifications to ensure that, even with prediction error, the product will meet the minimum specifications. In other words, the guardband is the amount of MHz by which the operating frequency defined in the datasheet has to be increased to ensure that no more than 0.1 parts per million (ppm) of false positives will be in production. To make an example, let considering that a machine learning model has been trained and a Guardbang G has been calculated. Supposing that the screening frequency is f_{screen}, the effect of the guardband is to increase the threshold for the pass/fail screening from f_{screen} to $f_{screen} + G$. The goal to achieve is to have the guardband as small as possible, as it affects the production yield. To define the guardband, an assumption on the error distribution has to be made: we assume that the error distribution of the model (that is the difference between measurements and predictions) is Gaussian. The definition of a guardband is strictly related to the Six-sigma (6σ) method: this is a method for quality control introduced by Motorola in 1986. The objective is to reduce the process output variation so that this will result in no more defect than $3.4ppm$. For a process with a given specification limit (that, in our case is the f_{screen} or the the maximum operating frequency of the devices that satisfies the production constraint), this results in 6 process standard deviations between the mean of the process and the specification limit. This result comes from the theory of Gaussian distributions, according to which a point belonging to a normal distribution has very high probability to fall into the 6 sigma range. Following this approach, we can compute the guardband on a test set with real frequency y and predicted frequency $\hat{y}$ as:

$$G = \mu_e + 6\sigma_e \tag{3.33}$$

in which μ_e and σ_e are the mean and the standard deviation of the error distribution, computed as $e = \hat{y} - y$, that is the difference between predictions and real frequency. It is possible to verify that G corresponds to a defective level below 0.001 ppm that is a stringent constraint. To stay below 0.1 ppm, it would be enough using 5.2σ. However, since the error distribution is not purely Gaussian but this is an approximation, it is common to also approximate the coefficient to 6. The problem of this approach is that the guardband so calculated fits only on

a test set and does not have statistical meaning for the whole production process, because of the test set is only a sample of the production distribution. For this reason, the correct way to compute a guardaband is done using *Bootstrap*. The empirical bootstrap is a statistical technique that permits to estimate a confidence interval of a statistic, on the basis of a sample of the population. In this case the statistic of interest is the guardband. It is based on sampling with replacement from the population, permitting to computing the guardband of the production process on the basis of the test set. The step to follow are:

- Sample n observation with replacement from the test set
- Compute the statistics on the sampled subset
- Repeat the previous two steps N times, with N sufficiently large.

The distribution obtained is called the *Bootstrap Distribution (D)*, that approximate the original distribution. The guardband is set as the upper bound of a confidence interval computed as:

$$CI = \mu_D \pm z_{1-\alpha/2}\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(D_i - \mu_D)^2} \tag{3.34}$$

where α is a confidence level and z is the quantile function of a normal distribution. The confidence level has to be set to 99.99999% to get a 0.1 ppm Guardband.

3.7.3 Classification

Since in the classification problem the label are discrete or categorical and not real number as in the regression task, new performances metrics have to be introduced. Considering a simple binary classification problem, in which there is only two classes and so the output can only be 0 or 1 (or -1,1, True or False, Positive or Negative ecc), four different cases may happens:

- An observations is predicted to belong to one class and it actually does: this is a True Positive (TP)
- An observation is predicted do NOT belong to a class and it actually does NOT belong to that class: this is a True Negative (TN)
- An observation is predicted to belongs to a class but it actually does NOT belong to that class: this is a False Positive (FP)
- An observation is predicted to NOT belong to a class and it actually does belong to that class: this is a False Negative (FN)

		True Classes		
		Positive	Negative	Total
Predicted Classes	Positive	$TP = a$	$FP = b$	$a + b$
	Negative	$FN = c$	$TN = d$	$c + d$
	Total	$a + c$	$b + d$	N

Table 3.3: Example of confusion matrix

These four possible outcomes can be seen in table called a confusion matrix, that count the number of TP, TN, FP and FN.

Accuracy

Accuracy is one of the most widely used metrics for evaluating classification models, applicable to both binary and multi-class classification tasks. It is defined as the ratio of correctly predicted instances to the total number of instances, as shown in the following formula:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.35}$$

However, accuracy is most meaningful when the dataset has balanced classes. For imbalanced datasets, accuracy can be misleading. For instance, if a dataset contains 99% of instances from the positive class and 1% from the negative class, a classifier that always predicts the positive class will have an accuracy of 99%, despite being ineffective at identifying the minority class. Therefore, additional metrics such as *Precision*, *Recall*, and *F1-score* are introduced to provide a more comprehensive evaluation.

Precision

Precision measures the proportion of correctly predicted positive instances out of all predicted positives. It is calculated as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.36}$$

In the earlier example of an imbalanced dataset, if the classifier never predicts the negative class, its precision for the negative class would be zero. Precision is particularly valuable in contexts where the cost of false positives is high, requiring a high degree of certainty in the positive predictions.

Recall

Recall, also known as sensitivity or true positive rate, quantifies the proportion of actual positive instances that are correctly identified by the model:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.37}$$

Recall is an essential metric when the objective is to identify as many positive instances as possible, such as in scenarios where the cost of false negatives is significant. For example, in medical diagnosis, detecting all potential cases of a disease (even with some false positives) may be crucial, or in quality control, ensuring all defective microcontrollers are captured is vital. Recall reaches its maximum value of 1 when the model predicts the positive class for all samples.

F1-score

The *F1-score* represents the harmonic mean of precision and recall, providing a single metric that balances both concerns:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.38}$$

The F1-score is especially useful when seeking a balance between precision and recall, particularly in cases where there is an uneven class distribution. A low precision or recall will result in a low F1-score, making it an effective measure for models where both metrics are important.

Weighted and Macro Averages

For multi-class classification problems, metrics such as Precision, Recall, and F1-score can be computed for each class separately. To assess overall model performance, these per-class metrics can be averaged. The *macro average* calculates the arithmetic mean by treating each class equally, regardless of the class frequency. In contrast, the *weighted average* takes into account the number of instances in each class, giving more weight to the majority classes in datasets with imbalanced class distributions.

3.8 Outlier Detection

Outlier detection is a critical task in many data science fields, as outliers often indicate abnormal conditions that can lead to significant performance degradation

in machine learning models. Examples of outliers include abnormal objects in images, intruders with malicious intentions within a system, and faults on production lines.

The effect of outliers and noise in the data measurement phase involves a shift in the underlying data distribution. If the data are affected by noise, the distributions of inputs $p(x)$, labels $p(y)$, and the joint distribution $p(x, y)$ may vary. Consequently, a machine learning model might learn different patterns compared to the information that can be retrieved from the data, particularly learning a different discrimination distribution $p(y|x)$ compared to that associated with the underlying processes.

Outlier detection has been categorized into three types [72].

Type 1 involves detecting outliers with no prior knowledge of the data and typically employs clustering approaches. Type 2 aims to model both normal and abnormal conditions, analogous to supervised classification, and requires pre-labeled data for normal and abnormal classes. Type 3 focuses solely on modeling normality; an algorithm autonomously learns to recognize abnormalities by defining the boundaries of normality. This approach is commonly referred to as novelty detection or novelty [72]. Our approach integrates aspects of Type 1 and Type 2 outlier detection, utilizing statistical tools to identify abnormal data while leveraging some pre-existing indications of outlierness.

Statistical approaches for outlier detection often use metrics such as the interquartile range (IQR) or the z-score.

The IQR method identifies outliers based on their position relative to a specific range, calculated using the first and third quartiles (Q_1 and Q_3) of the data distribution. The steps are: (1) calculate Q_1 and Q_3, and (2) compute the IQR as $Q_3 - Q_1$ and define the lower and upper bounds (lb and ub) as:

$$lb = Q_1 - 1.5 \cdot IQR \qquad ub = Q_3 + 1.5 \cdot IQR$$

Any data point outside the range $[lb, ub]$ is considered an outlier [86]. Note that the factor 1.5 can vary in different implementations.

The z-score measures how many standard deviations a sample is from the mean of a group of samples. The formula for the z-score is $z = \frac{x-\mu}{\sigma}$, where μ is the mean and σ is the standard deviation of the population. A z-score of 3 indicates that a sample is 3 standard deviations above the mean. For normally (i.e., Gaussian) distributed samples, the probability that a sample has a z-score below 3 is approximately 99.865%, meaning that there is only a 0.135% chance for a sample to exceed 3 standard deviations. Therefore, a z-score threshold of 3 will be used to differentiate between inliers and outliers in the experiments.

3.9 Imputation

Data are often imperfect, compromising of outliers and missing values. These may arise in both the feature space and the label space. In a single-label dataset, if a label is missing, the corresponding sample cannot be utilized in supervised learning algorithms and is typically discarded.

However, in cases where missing values occur in the feature space or in multi-label datasets, discarding samples may not be acceptable, particularly if the effort to collect those samples is substantial. A more effective approach is to recover missing values using mathematical procedures such as weighting or imputation [38], which infer missing values from the known data.

One type of imputation algorithm is the Univariate Imputation. This method imputes missing values in a column of the dataset by using only the non-missing values in that feature dimension. For example, missing values can be replaced with the mean, median, or another appropriately chosen value of that feature.

Multivariate imputation algorithms [41], on the other hand, use the entire set of available features to estimate missing values. In this approach, a missing value in one column is imputed based on information from all other columns. An example of a multivariate imputation method is the Iterative Imputer [75]. This method models each feature with missing values as a function of other features and uses this model for imputation.

The Iterative Imputer works as follows: each feature column with missing values is designated as the output y, and the remaining feature columns are treated as inputs X. A base regressor is fit on (X, y), and this regressor is then used to predict the missing values of y. This process is performed for each feature in a round-robin manner. The imputation procedure is repeated for a specified number of iterations, and the results from the final imputation round are used.

Chapter 4

Related Work of Machine Learning Applications in IC Testing and CAD

In recent years, the use of machine learning (ML) in integrated circuit (IC) design and testing has gained significant attention, with numerous data analytic methods based on ML being explored [145, 144]. While the design of electronic circuits involves a complex, multi-step process, ML can be utilized during the testing phase to enhance circuit quality both immediately after manufacturing and throughout the circuit's operational lifetime. The increasing complexity of IC designs necessitates the application of ML techniques to address scalability challenges in IC testing. Recent research has investigated various ML approaches across different stages of the production chain.

A comprehensive overview of recent advancements is provided in [5], summarizing contributions from three distinct research groups. The first group focused on interpretable ML in IC testing, which aims to provide insights into model predictions, unlike traditional "black-box" models, thereby aiding informed design decisions.

The second group examined deep learning techniques, particularly physics-informed convolutional neural networks, for electrostatic analysis. This approach aims to solve differential equations based on physical laws, enhancing the accuracy of electrostatic simulations.

The third group explored ML applications in emerging technologies, specifically in modeling degradation phenomena to study their impact at the chip level, even when defects manifest at the transistor level.

In [132], practical recommendations for ML applications in IC testing are detailed. One notable application is the "Alternate Test," which involves applying low-cost test stimuli and extracting alternate measurements to infer performance

implicitly. By selecting measurements highly correlated with performance metrics, variations in performance can be tracked through these measurements via regression models, simplifying the test process and reducing costs [141].

Historically, this approach has been extensively analyzed in analog domains rather than digital circuits. Notably, the correlation between indirect measurements, such as on-chip ring oscillator frequency values, and $F_{\max}$ has been a focal point in MCU performance screening [44].

Several studies have investigated the correlation between structural and functional $F_{\max}$ [120, 117, 44]. However, most existing works rely on simulated data [102] or have analyzed limited data samples (tens) from MCUs [50]. An ML-based approach to performance screening was first presented in [50], identifying the Gaussian Process as the most effective algorithm for learning from various feature sets. The study highlighted the negative impact of outliers on predictions and implemented a conformity check to remove them from the training set.

In [51], two lots of the same product, composed of 79 and 74 devices respectively, were compared. The models trained on one lot failed to predict devices from the other lot, potentially due to different packaging or limited sample sizes.

A more recent study correlated functional $F_{\max}$ with low-cost embedded sensors to measure the slack of selected paths accurately [117]. This method was tested on 300 OpenSPARCT2 SoC devices using transition-level simulations of two process variation scenarios.

Another application of ML to CAD is test compaction, which aims to reduce test costs by identifying and eliminating redundant information in the test set, thereby reducing the number and types of tests performed during production [39]. ML can be used to identify a subset of uncorrelated tests that can detect most faulty cases.

4.1 The state-of-the-art microntroller performances prediction

A limited number of works focused on MCU performance screening from alternate test. The main works are [50, 44]

In [50], authors explore the correlation between structural test measurements and functional $F_{\max}$, aiming to establish a reliable predictive model using data learning techniques.

The authors highlighted the significance of $F_{\max}$ as a critical performance metric, with functional $F_{\max}$ serving as the gold standard and structural $F_{\max}$ derived from various test patterns as a lower-cost alternative. The research investigates the correlation between functional $F_{\max}$ and structural $F_{\max}$ across five distinct

structural test pattern sets: complex transition patterns, simple transition patterns, Array Built-In Self-Test (BIST), complex path delay patterns, and simple path delay patterns. Notably, the study utilizes microprocessor samples operating above 1 GHz.

Despite the extensive analysis, the authors found no linear correlation between any of the five structural $F_{\max}$ results and the functional $F_{\max}$. However, by employing a binning strategy that categorizes frequency results into "slow" and "fast" bins, they identified that structural $F_{\max}$ based on complex path delay patterns exhibited the best correlation with functional $F_{\max}$. The methodology employed in the study includes a comprehensive evaluation of three types of structural delay measurements: flip-flop (FF)-based, pattern-based, and path-based datasets. The results indicate that the FF-based measurements yield the most accurate predictions, achieving near-zero error on training samples and superior performance on test samples. In contrast, path-based measurements were found to be the least effective for predicting $F_{\max}$, primarily due to their limited information coverage, as they only represent a small subset of the device's characteristics.

The authors introduce a data learning approach that incorporates a conformity check method, which selects a subset of conformal samples to enhance the reliability of the predictive model. This model consists of two components:

- **Conformal Model**: This model assesses whether a chip's Fmax is predictable based on its structural measurements. It employs a one-class learning approach to determine the conformity of samples, ensuring that only those samples that meet certain criteria are used for prediction. This helps in filtering out outliers and noisy data that could adversely affect the prediction accuracy.

- **Prediction Model**: The prediction model estimates Fmax based on the selected structural test measurements. Various learning methods were evaluated, including k-Nearest Neighbors (k-NN), Linear Regression, Ridge Regression (RG), Support Vector Regression (SVR), and Gaussian Process (GP) regression. The study concluded that the Gaussian Process regression method provided the best performance due to its ability to model complex relationships and quantify uncertainty in predictions.

In conclusion, the findings of the presented study underscore the importance of selecting appropriate structural test measurements and highlight the potential of data learning techniques in improving the predictability of $F_{\max}$. The insights gained from the research provide a foundational understanding for further exploration in microcontroller performance screening, particularly in optimizing structural test methodologies to achieve more reliable performance predictions.

[44] serves as a baseline for the research under analysis, which is being developed upon that findings by exploring additional methodologies and techniques for enhancing microcontroller performance screening.

Authors proposed a novel methodology for performance screening based on SMONs and ML techniques. This involves a two-step measurement process. Initially, SMONs are measured while the die is still on the wafer, allowing for high-precision readouts. Following packaging, functional patterns are tested on the microcontroller to establish reference labels for machine learning training. This dual approach ensures that the performance data collected is both accurate and relevant to real-world applications.

The results presented in the study indicate that the proposed methodology significantly enhances the ability to predict microcontroller performance. The authors report that their approach can achieve a false positive rate of less than 0.1 parts per million (ppm), which is crucial for maintaining the high quality required in automotive applications. This level of accuracy is achieved by employing advanced machine learning techniques that analyze the multidimensional data from SMONs, allowing for reduced margins while still effectively identifying under-performing devices. Experiments was conducted on a dataset composed by 3616 labelled devices from 26 corner wafers. This dataset will be either used in this work for the development of advanced machine learning models.

In [44], regression models were chosen to cover the wide amount of existing models in machine learning realm. The choice of the estimator was proven not to be critical, as many estimators show similar results. Both Linear and Non Linear models were evaluated. In particular, authors experimented on Linear Regression, Lasso, Ridge, and ElasticNet regression and SVR with radial basis function (RBF) kernel. Moreover, tree-based ensemble models (Random Forests and Extra Trees) performed similar to SVR or slightly better.

The study also highlights the importance of addressing the slow-performing devices in the left tail of the performance distribution. By focusing on these devices, the proposed methodology aims to minimize yield loss and ensure that only devices meeting the stringent performance criteria are shipped to customers.

As outlined, in [44] the foundational techniques for the ML-based MCU performance screening framework are established. These will be further elaborated in the subsequent chapters.

Chapter 5

ML-based Performance Screening Foundation

This chapter establishes the foundation for ML-based performance screening, specifically focusing on the analysis of initial devices from split lots of a particular MCU product, denoted as $A1$. It begins with an overview of the data collection phase for dataset $A1$ (section 5.1) and discusses the challenges encountered during the labeling process.

The chapter then presents preliminary findings from earlier studies on the same dataset, as reported in [44] (section 5.2). Building on these initial results, it progresses to the development of *advanced* models by integrating state-of-the-art techniques detailed in section 5.2. These models employ a combination of feature preprocessing, linear modeling, and multi-target learning methods (sections 5.2.1 and 5.3).

Subsequently, the chapter explores the optimization of the characterization process and dataset labeling using active learning strategies (section 5.4). The outcomes of these analyses, along with comparisons to state-of-the-art methods, are presented in section 5.5. Additionally, it addresses challenges related to the dataset, such as the presence of outliers and missing values, and proposes strategies for managing these issues (section 5.6). The overall proposed workflow is outlined in section 5.7. Finally, section 5.8 concludes the chapter.

5.1 Data Collection

The collection of the data can be separated into two parts: (i) The SMON readout of the chip (as described in section 2.3), and (ii) the measurement of the F_{max}, as described in section 2.5. Devices that have only the SMON readouts are the *unlabeled* devices, while a *labeled* device has the SMONs readout and

goes through a separate characterization process with the functional patterns. SMONs readouts lead the *features* for the ML training process, while the $F_{\max}$ measurements give the *reference labels*.

The measurements for each sample, both for features and labels, may be subject to measurement uncertainty and noise.

Besides label uncertainty described in section 2.5, other forms of variation are (1) the process variation in the CMOS manufacturing and (2) devices with a defect. Every CMOS manufacturing has a particular process variation which usually follows a Gaussian distribution [98]. This means a vast amount of devices are within the expected process variation. However, some devices are in the tail region of the distribution. Those devices can be determined as outliers. Devices with physical or local defects are identified during the production phase and, considered outliers, are screened out. There are test methods in the production flow (e.g., based on Stuck-At, Transition, or Path-Delay testing) trying to identify such defective devices [104]. However, some MCUs with certain defects might escape the test procedure, or the tests are still in the engineering phase and not mature enough.

Defective labeled devices that have escaped production filters can be identified and screened out by calculating the so-called *Outlier Index*. For each device and each functional test pattern, if the measured label deviates from the median of the labels in its wafer (*wafer-median*) by more than 2.5 standard deviations, it can be classified as an *outlier*. In some instances, a device may be classified as an outlier for each functional test pattern, whereas in others it may be classified as an outlier for only one or a few patterns (i.e., the remaining labels fall within the expected distributions). The *Outlier Index* is computed as:

$$\text{Outlier Index} = \frac{n}{x} \tag{5.1}$$

where n is the number of test patterns where the device is identified as an outlier, and x is the total number of test patterns. Thus, the Outlier Index ranges from 0 to 1. In the baseline approach, a device is removed from the training set if it is detected as an outlier for at least one functional test pattern. This conservative approach ensures high-quality training data but may be overly stringent.

The boxplot in fig. 5.1 illustrates how $F_{\max}$ varies for a set of devices from a wafer of a particular MCU product (A). One device is clearly below the threshold value indicated in the figure for several functional test patterns, while other devices are just below the whiskers.

As mentioned in section 2.5, the labeling procedure is performed mostly manually, and forms of uncertainty are possible. Some mechanisms are thus integrated to make the label acquisition more robust. Each determination of the $F_{\max}$ is

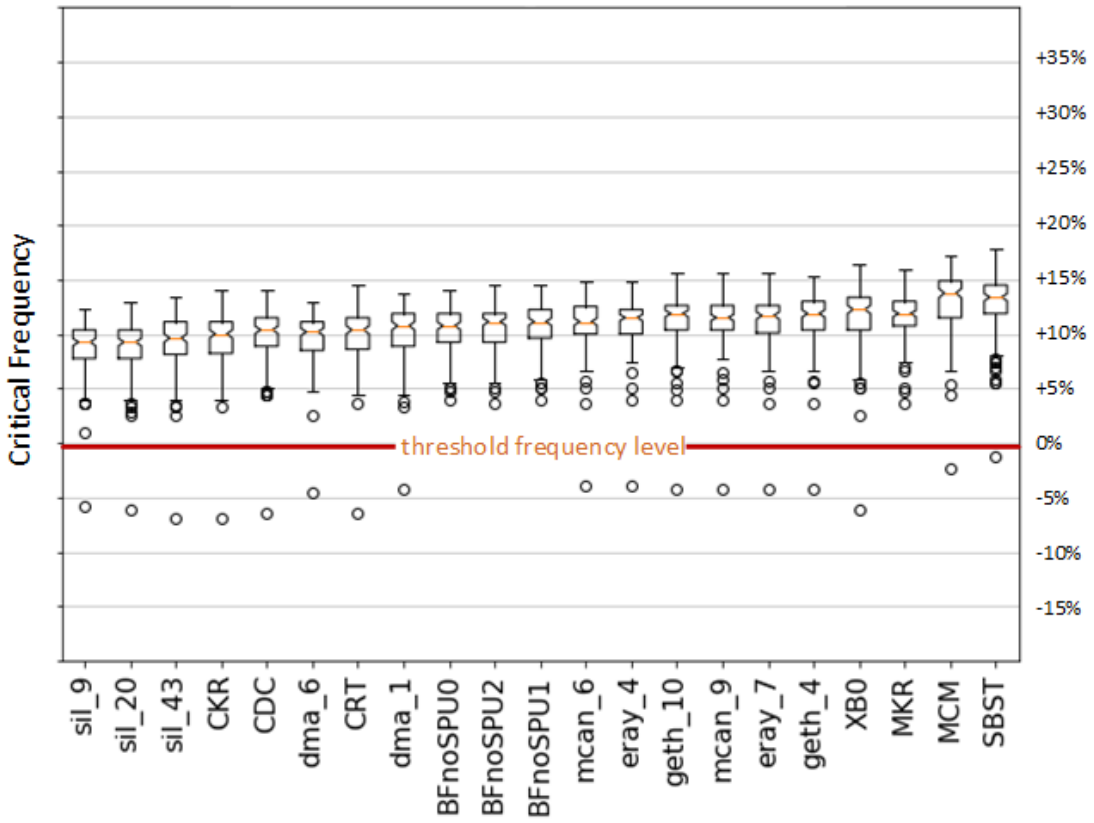


Figure 5.1: An overview of functional patterns and their critical fail frequency with the threshold frequency level of the data-sheet. Some outliers are visible over all applied patterns [44]. The threshold is 0%.

repeated multiple times (five in our case study) for every device and for every pattern; the median value is calculated and determined as the final $F_{\max}$ value. If a device has a high variance between the various measurements, the label is treated as an anomaly and discarded, introducing a *missing value* in the dataset. The concept of *high variance* is at the discretion of test engineers' domain experts, who decide if to remove or not the label measurement from the dataset. An example of this can be see in fig. 5.2

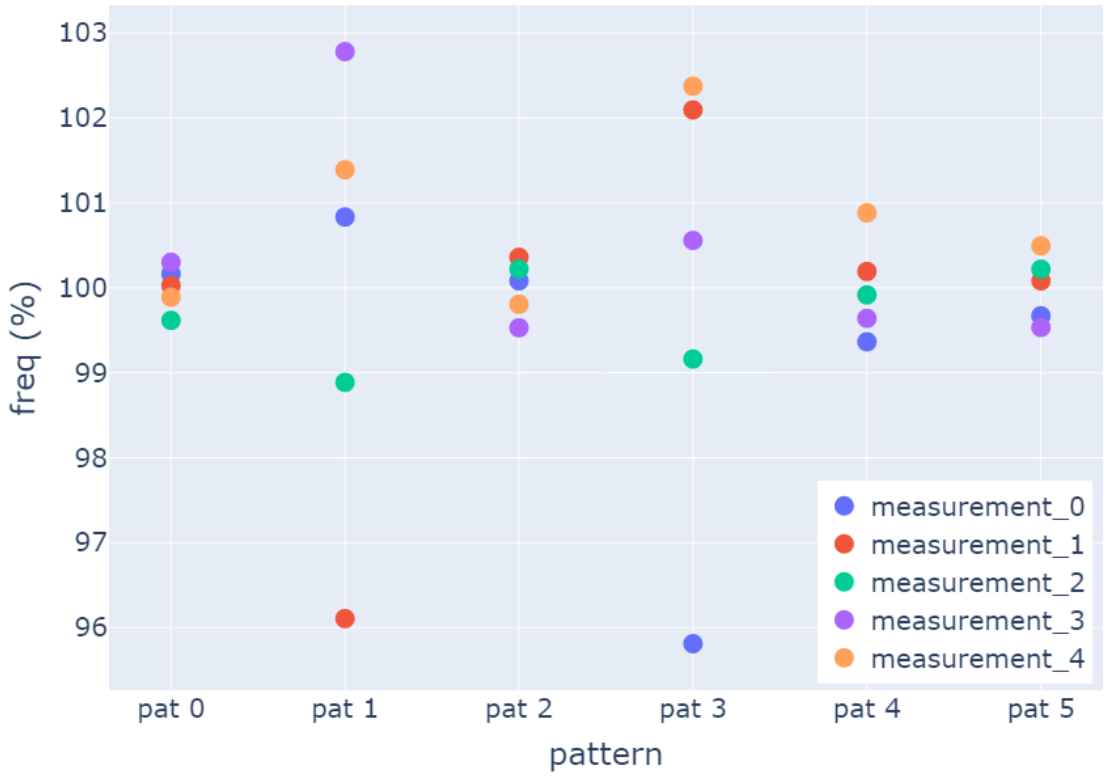


Figure 5.2: Graphical view of a toy-example of label measurements for a device. Five patterns are measured and five measurements are taken for each pattern. In this example, the frequency for patterns 2 and 3 would be discarded from the dataset because of the high variance.

In the most conservative approach, only samples with an outlier index equal to 0 are selected for the learning phase, that is, if a device results to be an outlier in just one test, it is marked as an outlier and removed from the training set. In addition, only the devices for which all the test pattern measurements are available are considered valid devices to be inserted in the training set. If a sample lacks one or more label measurements it is discarded from the training set. Even if removing a sample that is an outlier may increase the quality of training data, it

is also true that if only one (or a limited number) of the test patterns is faulty, it does not necessarily imply that the entire device is defective; however, in the classic approach, all captured labels for that device are discarded.

This choice, which is the most pessimistic, leads to the least uncertain possible dataset, removing both uncertainty in the labeling process and possible defective devices that have escaped production screening.

However, the reduction in the available sample is huge: after having removed samples with at least one missing value due to the high variance in the repeated F_{max} measurements and outliers, less than 50% of the samples may be available for successive model training and analyses. This choice would be the baseline for preliminary experiments, with the goal of enhance state-of-the-art model for MCU performance prediction(section 5.5). Specific methods are thus needed to cope with such a dataset. These will be developed in (section 5.6)

5.2 Machine Learning models

Preliminary machine learning models, building on previous work [44], predominantly utilize Linear Models (such as Ridge Regression), SVR, and Tree-based models. The main results from training models on a dataset composed of 2,048 samples derived from 26 split-lot wafers with 27 SMONs are highlighted. The relatively low prediction error achieved by Linear Models (approximately 2.4% nRMSE) indicates that the correlation between SMONs and the target label (P_{min}) can be well approximated by linear relationship. It is good practice to use linear models as a baseline: they provide a straightforward point of comparison for more complex models. By starting with linear models, we can establish a clear understanding of the fundamental relationships within the data. This baseline can then be used to evaluate the improvements offered by more sophisticated, non-linear models.

However, the even lower prediction error obtained by non-linear models (around 1.6% nRMSE achieved by both SVR and Tree-based models) suggests that linear models might overlook some interactions between SMONs, indicating that the relationship between SMONs and F_{max} may not be strictly linear. This suggests that the relationship between input and output is not highly complex, even though a purely linear relationship does not perfectly fit the task.

Despite this, linear models are often preferred due to their simplicity, interpretability, and computational efficiency. These characteristics are valuable, especially in the initial stages of model development. They offer insights into the underlying data structure and can help identify potential areas where non-linear models might perform better. Even if linear models may not capture all interactions, their role as a baseline is crucial for benchmarking and guiding the development of more

advanced models.

While the relationship between SMONs might not be strictly linear, a transformation of SMONs could project them into an alternate space where the relationship with $F_{\max}$ is linear. Therefore, an effective preprocessing step could involve transforming the raw SMON frequencies, enabling the use of linear models to infer $F_{\max}$ in this transformed space. This strategy is described and utilized in the following subsections.

5.2.1 Polynomial Transformation

To address the possible oversimplicity of linear models, nonlinear feature transformations can be applied to project the input feature space into a higher-dimensional space. In this transformed space, it is more likely that the input-output relationship becomes linear, thereby enabling the use of linear regression models. The transformation employed in this context is the polynomial transformation, which generates polynomial combinations of the input features up to a specified degree. For example, the degree-2 polynomial features of a 2-dimensional input sample (a, b) are given by:

$$\Phi(a, b) = (1, a, b, a^2, ab, b^2) \tag{5.2}$$

As an example, 27 SMONs generate 406 features with a 2-degree transformation and 4060 features with a 3-degree transformation.

5.2.2 Feature Space reduction

Since the number of polynomial features may be relatively high, applying dimensionality reduction techniques can be beneficial for reducing model complexity and improving performance. Looking at the correlation among SMONs can help in selecting only those features that significantly contribute to the model's quality. Features with strong linear dependencies may be redundant or even detrimental, potentially causing overfitting and increasing model size without improving performance.

For feature space reduction, the Pearson Correlation [121] heatmap among SMONs is useful to reveal correlation among SMONs. If we encounter highly linear correlations among groups of SMONs, it is possible to compact information by either I) selecting the features most interesting for the task or II) creating new features by combining existing ones. The first strategy is at the base of Feature Selection approaches, while the second is the principle of Feature Extraction.

Two dimensionality reduction approaches are evaluated: Principal Component Analysis (PCA) as a feature extraction method and Univariate Feature Selection

(UFS) as a feature selection method. In the PCA approach, we extracted the first k Principal Components (PCs) and applied regression on this new feature space. In contrast, UFS selects the best k features based on univariate statistical tests, eliminating the rest. PCA creates new features, while UFS simply discards less relevant ones.

Alternatively, it is possible to combine both feature space reduction and transformation in a single step using Kernel PCA (KPCA) [146, 122]. KPCA extends PCA by achieving non-linear dimensionality reduction through kernels, which map data into a higher-dimensional space to capture more complex patterns. KPCA constructs the kernel matrix K from the training dataset using a kernel function k (i.e., $K_{i,j} = k(x_i, x_j)$ for each pair of samples (x_i, x_j)) and then computes kernel principal components, similar to PCA.

5.3 Multi Target learning

As stated in section 2.5, labeling the devices involves computing performance on more than one pattern. Usually, this is done for tens of testing programs. Thus, the dataset is composed of multiple labels. Even if the testing programs are different and aim to stimulate different parts of the devices, the final labels are highly correlated. This observation justifies the use of multi-label learning strategies. Predicting one label can be useful in predicting the others, which is the fundamental concept behind multi-label regression.

This approach leverages the correlation in the label space to predict the minimum maximum operating frequency (i.e., the artificial target P_{min}). Two methods are used to deploy this concept: the *Regressor Chain*, based on the idea of chaining single-target models and creating a list of regressors in which each takes advantage of the predictions made by its predecessors in the list; and the *Multi-Target regressor stacking*, which is a two-stage approach (briefly explained in Chapter 3.3). The two multi-target regression approaches are illustrated in Figure 5.4.

5.4 Active Learning

As detailed in section 2.5, the process of acquiring labeled data predominantly involves manual efforts, rendering it both challenging and resource-intensive. The manual nature of this labeling process results in slow and costly data acquisition, exacerbating the difficulties associated with managing and optimizing datasets. Additionally, the presence of outliers and missing values in the data, as highlighted in section 5.1, can adversely affect model performance if not adequately addressed. Consequently, the optimization of the training set is essential for enhancing the

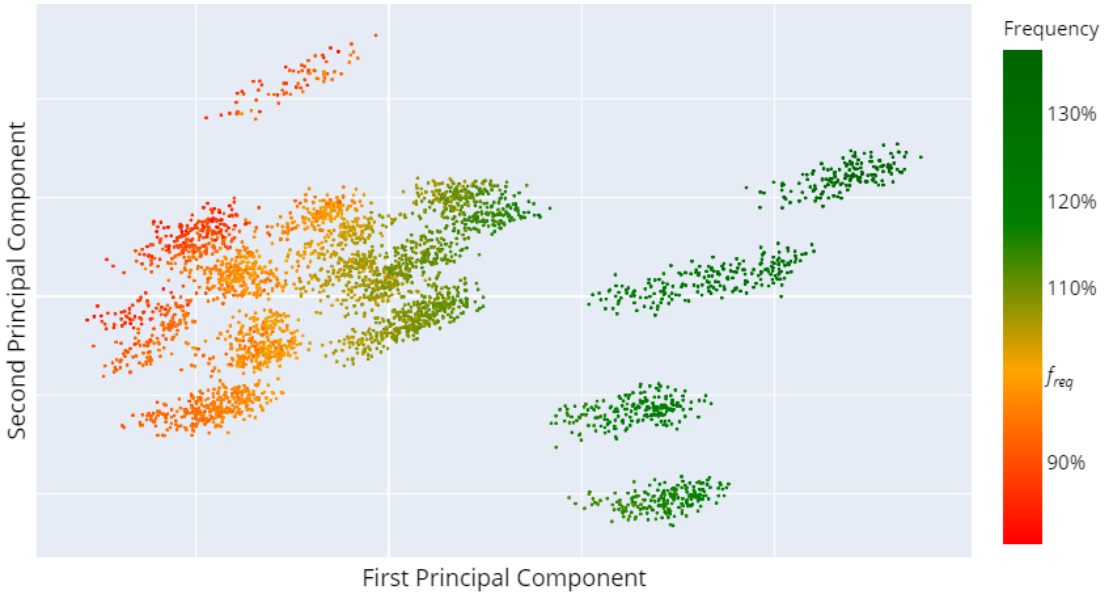


Figure 5.3: Two-dimensional PCA representation of corner-lot devices used for initial model characterization. The color scale indicates the actual frequency of the devices in Hz relative to the specification f_{req}.

efficiency and efficacy of machine learning models.

Active learning offers a strategic methodology for addressing these issues by refining the selection of training samples. Rather than arbitrarily choosing samples for labeling, active learning aims to identify those samples that will most significantly benefit the model's performance. This method is grounded in the notion that not all data points contribute equally to the model's improvement; rather, certain samples provide more critical information that can augment the model's generalization capabilities to novel, unseen data.

Implementing active learning allows for the efficient selection of the most informative samples for labeling. This focused strategy not only aids in constructing models with superior generalization capabilities but also reduces the overall labeling workload and accelerates the training process. In the realm of MCU performance screening, active learning thus addresses issues related to data scarcity and quality, ultimately leading to more effective and efficient performance predictions.

The active learning selection strategies employed are outlined in section 3.2. These strategies include:

- Local Outlier Factor (LOF)
- Query by Committee (QBC)
- Euclidean Distance (EDD)

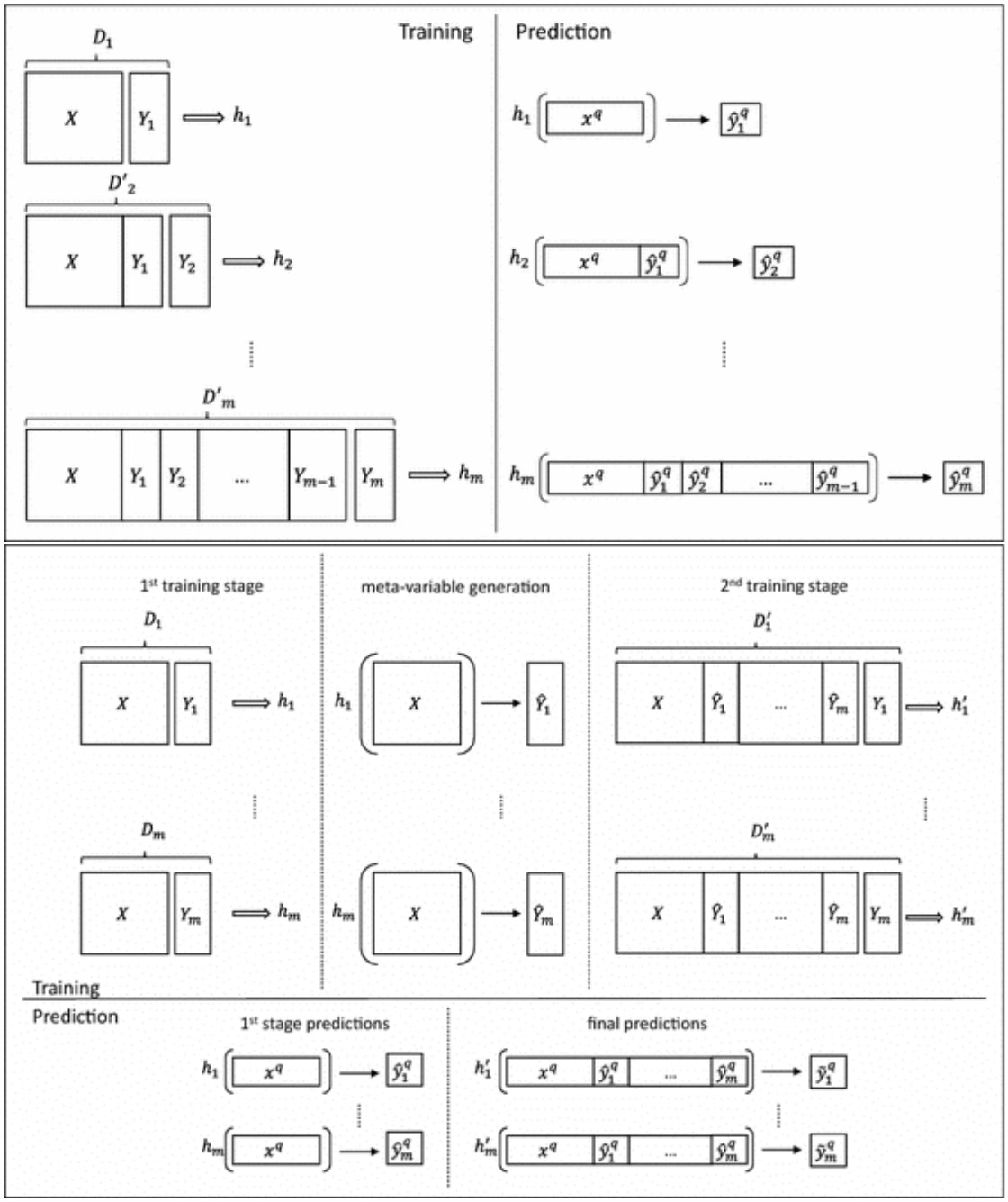


Figure 5.4: Graphical illustration of Regressor Chain (above) and 2-Stage Regressor (below) training and prediction procedures [130]

- Greedy Sampling on the Input (GSx)

- Greedy Sampling on the Output (GSy)
- Improved Greedy Sampling (iGS)
- Hausdorff Distance (HD)

Preliminary investigations into active learning selection strategies have primarily focused on the models employed in the (QBC) and *GSy-iGS* methodologies. These strategies involve fine-tuning hyperparameters to optimize performance. Typically, models demonstrating minimal prediction error on validation datasets are considered strong candidates. Furthermore, to ensure committee diversity in QBC, a varied set of models is chosen to represent different data characteristics. These models are then trained on a uniform training dataset. The models evaluated for the *QBC* approach in this study include:

- Ridge Regressor with Polynomial Input Transformation (Poly Ridge)
- Support Vector Regression (SVR)
- Random Forest
- Gaussian Process Regressor

For the GSy and iGS approaches, Poly Ridge has been utilized as a baseline model; however, alternative effective baseline models are also considered suitable candidates. For the experiments conducted, the parameter $K$of LOF strategy (section 3.2) was set to 10, and the Euclidean distance (i.e., Minkowski distance with $p = 2$) was employed for distance calculations.

5.5 Experiments and Results

Preliminary data and models are sourced from an Infineon automotive MCU, fabricated using mature CMOS technology. The dataset comprises 4,039 samples derived from 26 split-lot wafers, covering a broad spectrum of technological behavior for the MCU product. Each device features 27 SMONs, with labels measured across 10 functional test patterns. Additionally, an artificial label, representing the minimum operating frequency among these patterns for each device (P_{min} or *PERF*, as detailed in section 2.5), serves as the final target in the performance screening procedure, identifying the critical pattern for each device.

The dataset can be visualized through a 2-dimensional PCA decomposition of the 27 SMONs, as illustrated in fig. 6.2, where the 26 wafers are distinctly differentiated.

After addressing missing values and outliers, the dataset is reduced to 2,407 samples, which are then split into training and test sets in a 75-25% ratio. This split is stratified by wafer to maintain the proportion of samples per wafer in both subsets. Consequently, the baseline training set comprises 1,805 full-row samples, representing less than 53% of the available devices, with the remaining 602 samples allocated to the test set for validating and evaluating the various experiments and ML models.

The removal of outliers and samples with missing values leads to the exclusion of 1,632 samples from the training set. Among these, 423 devices exhibit an *Outlier Index* greater than zero, indicating deviation from the wafer median (see section 5.1), while 1,209 samples contain missing values due to high measurement variance.

For the development of advanced models, the baseline training set, free of outliers and missing values, is utilized.

All 27 features are standardized as a crucial preprocessing step in regression tasks, particularly when models incorporate polynomial or interaction terms, which can induce multicollinearity and obscure significant variables. Standardization effectively mitigates multicollinearity issues and ensures comparability among features by scaling the data to a common range without altering the inherent differences. The Standard Scaler method is applied, which involves centering the data by removing the mean and scaling to unit variance. The standardization process is performed using the following formula:

$$\hat{X}_i = \frac{X_i - \mu_i}{\sigma_i} \tag{5.3}$$

where μ_i represents the mean and σ_i denotes the standard deviation of the i-th feature.

5.5.1 Basic Models

Results from basic models in [44] are presented. Both Linear and Non-Linear models were taken into account, as depicted in section 4.1.

Table 5.2 reports average results (among all 10 patterns). The table reports normalized Mean Absolute Error (nMAE), normalized Root Mean Squared Error (nRMSE), and the Guardband. For the guardand estimation with bootstrapping method, G was reported using the target False Positive rate. Tree-based ensemble models and SVR show the best results. According to the table, using such algorithms, it is possible to apply a performance screening at around 113% f_{nom} during production, while using less accurate models (e.g., Linear Regression) would require screening at 120% f_{nom}.

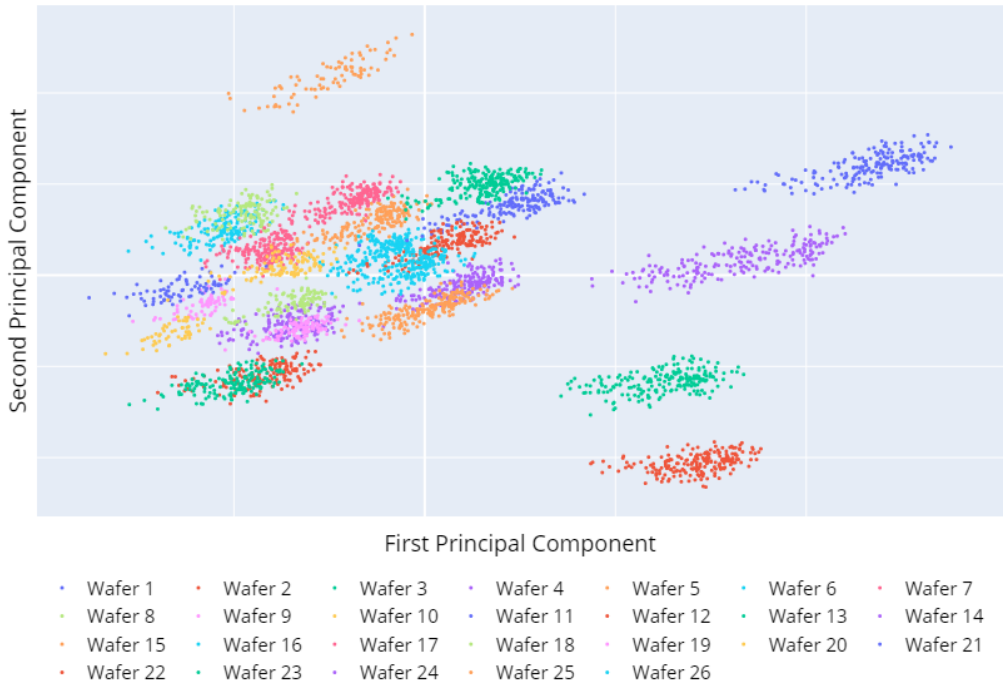


Figure 5.5: PCA-based two-dimensional representation of the corner-lot devices used for the initial model's characterization, colored by wafer.

Table 5.1: Normalized root mean squared error (nRMSE) on the test set using basic models

Algorithm	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Lin. Regr.	2.4%	2.1%	2.6%	2.4%	1.9%	2.9%	2.4%	2.5%	2.5%	2.5%
Lasso	2.4%	2.1%	2.6%	2.4%	1.9%	2.9%	2.4%	2.5%	2.5%	2.5%
Ridge	2.4%	2.1%	2.6%	2.4%	1.9%	2.9%	2.4%	2.5%	2.5%	2.5%
ElasticNet	2.4%	2.1%	2.6%	2.4%	1.9%	2.9%	2.4%	2.5%	2.5%	2.5%
SVR (rbf)	1.6%	1.5%	1.6%	1.6%	1.3%	1.9%	1.5%	1.6%	1.6%	1.5%
Rand. For.	1.6%	1.5%	1.6%	1.7%	1.3%	1.9%	1.5%	1.6%	1.6%	1.6%
Extra Trees	1.6%	1.5%	1.6%	1.6%	1.3%	1.9%	1.5%	1.6%	1.6%	1.5%

5.5.2 Advanced Models

The initial objective is to develop models that outperform the baseline linear models presented in [44]. To this end, features are transformed using a 2-degree polynomial expansion of the 27 SMONs, resulting in 406 new features. A higher-degree transformation, such as degree 3, would significantly increase the feature count to 4,060. The 2-degree transformation was selected due to its effectiveness in

Table 5.2: Basic models scores (mean values on 10 patterns)

Algorithm	Test set			Production (Bootstr.)	
	nMAE	nRMSE	G	G (0.3 ppm)	G (0.1 ppm)
Lin. Regr.	1.8%	2.4%	15.9%	19.3%	19.7%
Lasso	1.8%	2.4%	15.9%	19.3%	19.8%
Ridge	1.8%	2.4%	15.9%	19.2%	19.7%
ElasticNet	1.8%	2.4%	15.9%	19.3%	19.8%
SVR (RBF)	1.1%	1.6%	10.4%	12.5%	12.8%
Rand. For.	1.2%	1.6%	10.4%	12.5%	12.8%
Extra Trees	1.2%	1.6%	10.2%	12.2%	12.5%

minimizing prediction error while avoiding overfitting, and its manageable number of features relative to the available labeled samples.

The analysis encompasses both linear and non-linear approaches. Linear models with L1-L2 regularization—specifically Ridge [73, 111], Lasso [134], and ElasticNet [63, 161]—are employed, alongside non-linear methods such as kernel SVR [11] and tree-based models [**Breiman2001**, 152]. This variety ensures broad coverage of commonly used models in machine learning tools.

Models were trained using 5-fold Cross Validation (5-fold CV), and Grid Search was utilized to determine the optimal hyperparameters from a predefined grid, aimed at minimizing the generalization error.

Given the substantial number of features generated through polynomial transformation, feature space reduction becomes imperative, as outlined in section 5.2.2. Initial analysis involved computing the Spearman Correlation [128] heatmap (fig. 5.6), which identified that many SMONs exhibited strong correlations, with each SMON correlating almost perfectly with 12 to 14 others. This necessitated dimensionality reduction to address potential overfitting issues.

Two approaches for feature reduction were employed: univariate feature selection (UFS) and Principal Component Analysis (PCA) (section 5.2.2).

Table 5.3 summarizes the outcomes of these analyses. The first two columns detail the number of features (k) utilized and the explained variance (EV) ratio [77] achieved through PCA. It was observed that more than 99% of the variance could be accounted for with just 3 principal components. The subsequent columns display the mean normalized Root Mean Square Error (nRMSE) across 10 patterns computed on the test set, using three representative algorithms: Polynomial Ridge, SVR, and Extra Trees.

The results demonstrate that the PCA-based approach generally surpassed

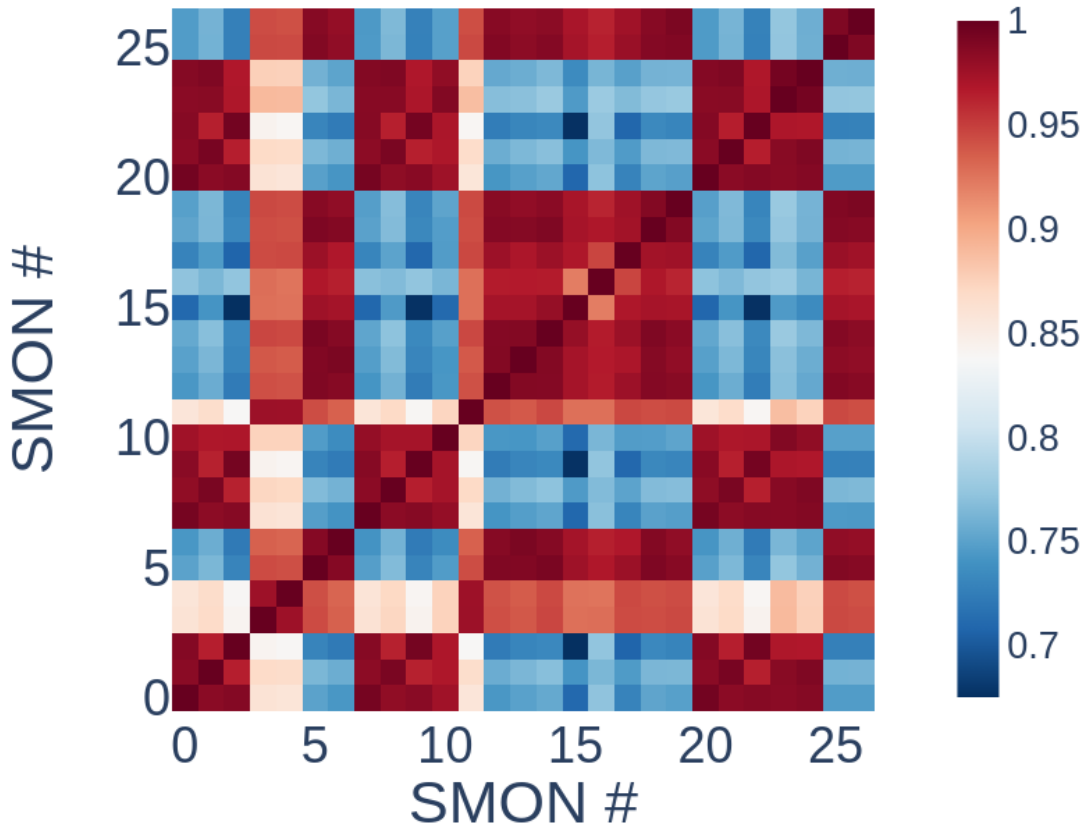


Figure 5.6: Spearman Rank Correlation matrices for product A, with 27 SMONs. Clusters of features can be identified.

UFS for the same k. Additionally, UFS error rates decreased as k increased, whereas fewer principal components occasionally produced superior results compared to using all original features (e.g., SVR with RBF kernel performed optimally with only 4 PCs and exhibited overfitting with additional PCs). In contrast, tree-based models were less sensitive to this behavior.

Incorporating the feature selection step with PCA prior to polynomial transformation ensures that the performance of linear models aligns more closely with the results presented in Table 5.4.

The final results utilizing the advanced models are presented in table 5.4. This table compares the performance of the proposed advanced models with state-of-the-art approaches [44]. For the advanced Ridge model, PCA with 14 components and polynomial transformation was employed, resulting in a total of 120 features. The SVR model utilized PCA with 4 principal components and an RBF kernel. For the advanced Extra Trees model, PCA with 14 principal components was applied.

The performance screening in production was improved from 120% f_{nom} (using

the basic versions of the models, section 5.5) to 113% f_{nom} with the advanced models. Due to their efficient inference, minimal memory usage, and straightforward implementation in languages suitable for embedded systems (such as C and C++), these advanced models are recommended for on-board predictions. Notably, the Ridge Regressor is designated as the baseline model for subsequent experiments in the following chapters.

5.5.3 Multi-Target Regression

The Pearson correlation matrix for the labels was computed and is presented in table 5.5. It is evident that the labels exhibit extremely high correlation with one another.

The Regressor Chain methodology was implemented across various common regression models, including those mentioned in (section 3.6). Comparisons were made between chained and non-chained models. Each regression model underwent hyperparameter optimization through Grid Search with 5-fold Cross-Validation (CV), executed over 20 training-test splits in a nested CV framework to ensure an unbiased estimate of generalization error [109]. The inner CV loop was responsible for model selection and hyperparameter tuning, while the outer loop estimated generalization performance.

A paired t-test was conducted on the 20 results obtained from the outer CV loop to assess whether observed performance differences between models were statistically significant. Each model was integrated into a pipeline with the following preprocessing steps:

- **Ridge, Lasso, ElasticNet, OMP**: Application of Standard Scaler, PCA with 14 components, and polynomial transformation of degree 2 (based on prior results).
- **SVR**: Standard Scaler, PCA with both 14 and 4 components, and RBF kernel (based on prior results).
- **Decision Tree, Extra Trees, Random Forest**: Standard Scaler, PCA with 14 components.
- **Ridge, Lasso, ElasticNet with KPCA**: PCA with 14 components combined with RBF and Polynomial (degree 2) kernels.

[1]Similar results can be obtained with other types of linear and tree-based models. Linear models benefit from input transformation, while tree-based models seem not to.

Table 5.3: Comparison between PCA and UFS (mean nRMSE on 10 patterns, advanced independent models)

#Features	EV ratio	PCA (nRMSE)			UFS (nRMSE)		
		Rigde[1] (poly)	SVR (RBF)	E.Trees	Rigde (poly)	SVR (RBF)	E.Trees[1]
1	87.61%	2.84%	2.57%	2.53%	3.15%	2.93%	2.92%
2	98.90%	1.81%	1.71%	1.70%	3.06%	2.74%	2.75%
3	99.24%	1.62%	1.54%	1.60%	3.02%	2.66%	2.70%
4	99.47%	1.61%	1.54%	1.59%	1.86%	1.84%	1.83%
5	99.52%	1.60%	1.54%	1.59%	1.81%	1.80%	1.81%
6	99.57%	1.60%	1.54%	1.59%	1.79%	1.79%	1.80%
7	99.62%	1.60%	1.54%	1.60%	1.79%	1.78%	1.80%
8	99.67%	1.60%	1.55%	1.60%	1.79%	1.78%	1.79%
9	99.71%	1.60%	1.55%	1.60%	1.75%	1.72%	1.76%
10	99.74%	1.59%	1.54%	1.60%	1.74%	1.71%	1.75%
11	99.77%	1.59%	1.54%	1.59%	1.73%	1.70%	1.74%
12	99.79%	1.58%	1.55%	1.60%	1.71%	1.68%	1.72%
13	99.81%	1.58%	1.55%	1.59%	1.68%	1.65%	1.68%
14	99.83%	1.58%	1.55%	1.59%	1.68%	1.64%	1.67%
15	99.85%	1.58%	1.55%	1.60%	1.68%	1.64%	1.66%
16	99.86%	1.58%	1.55%	1.60%	1.63%	1.58%	1.63%
17	99.88%	1.58%	1.56%	1.60%	1.62%	1.57%	1.60%
18	99.89%	1.58%	1.56%	1.59%	1.62%	1.57%	1.60%
19	99.90%	1.58%	1.56%	1.60%	1.62%	1.57%	1.59%
20	99.92%	1.58%	1.56%	1.60%	1.61%	1.56%	1.59%
21	99.93%	1.58%	1.56%	1.59%	1.60%	1.56%	1.57%
22	99.94%	1.58%	1.56%	1.61%	1.60%	1.56%	1.57%
23	99.96%	1.58%	1.56%	1.61%	1.60%	1.56%	1.58%
24	99.97%	1.58%	1.56%	1.60%	1.60%	1.56%	1.57%
25	99.98%	1.58%	1.56%	1.60%	1.60%	1.56%	1.56%
26	99.99%	1.59%	1.56%	1.61%	1.59%	1.57%	1.56%
27	100.00%	1.59%	1.56%	1.61%	1.59%	1.56%	1.55%

Table 5.4: Advanced models vs Base models scores (mean values on 10 patterns, independent target)

Algorithm	Base Models			Advanced Models		
	nMAE	nRMSE	G	nMAE	nRMSE	G
Ridge, 14 PC[1]	1.8%	2.4%	15.9%	1.19%	1.58%	10.30%
SVR, 4 PC	1.1%	1.6%	10.4%	1.12%	1.54%	10.22%
Extra Trees[1]	1.2%	1.6%	10.2%	1.15%	1.55%	10.15%

Table 5.5: Pearson correlation between Labels

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
P1	1	0.99	0.99	0.99	0.98	0.99	0.99	0.99	0.99	0.99
P2	0.99	1	0.99	0.99	0.99	0.98	0.99	0.99	0.99	0.99
P3	0.99	0.99	1	0.99	0.99	0.98	0.99	0.99	0.99	0.99
P4	0.99	0.99	0.99	1	0.99	0.98	0.99	0.99	0.99	0.99
P5	0.99	0.99	0.99	0.99	1	0.98	0.99	0.99	0.99	0.99
P6	0.98	0.98	0.98	0.98	0.98	1	0.98	0.98	0.98	0.98
P7	0.99	0.99	0.99	0.99	0.99	0.98	1	0.99	0.99	0.99
P8	0.99	0.99	0.99	0.99	0.99	0.99	0.99	1	1	1
P9	0.99	0.99	0.99	0.99	0.99	0.98	0.99	1	1	1
P10	0.99	0.99	0.99	0.99	0.99	0.99	0.99	1	1	1

- **Kernel Ridge with RBF kernel**: Ridge Regression employing the kernel trick [143].
- **Multivariate Adaptive Regression Splines (MARS)**: Applied with both the full feature set and with 14 principal components [63].

The regressor chain was organized in the following sequence:

$$C = (P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P_{\text{min}})$$

The results are detailed in Table 5.6. The regressor chain slightly outperforms the non-chained models for Ridge, Lasso, OMP, and ElasticNet, as evidenced by reductions in all computed error metrics and G. For linear models, G falls below 10% with nRMSE approaching 1.5%. In contrast, tree-based regressors performed better in non-chained configurations, potentially due to overfitting as the ratio $\frac{N_{\text{samples}}}{N_{\text{features}}}$ decreases with each added feature in the chain. No significant improvement was observed for SVR-4 in the chained version. For SVR-14, no conclusive evidence was

found: a t-test on the 20 splits yielded a p-value greater than 5%, indicating that performance differences may be attributed to the variability in training and test sets. Polynomial-PCA linear regression models and MARS models demonstrated slight improvements with the chained configuration.

The KPCA approach resulted in higher prediction errors compared to separate feature reduction and transformation steps, as shown in table 5.6. However, it achieved a final feature space dimension of 14, in contrast to 120 achieved by applying PCA with 14 components followed by a polynomial transformation of degree 2.

5.5.4 Discussion on ML models

The existing performance prediction models, as detailed in [44] and referred to as basic ML models, have been successfully enhanced through various techniques. Dimensionality reduction methods, such as PCA, prove effective in minimizing the number of features that predictive machine learning models need to process. PCA condenses the information into new features, leading to more streamlined models without significant loss of information. Additionally, polynomial transformations enable classical linear regression algorithms to address non-linear relationships, thereby increasing their applicability to the problem at hand.

Experimental results demonstrate that satisfactory performance, with a normalized Root Mean Square Error (nRMSE) of approximately 1.5%, can be achieved across a range of models. However, linear models are favored due to their interpretability, as they assign coefficients to each feature, providing valuable diagnostic information on feature importance. While kernel methods also deliver strong performance and are consistent with other approaches, they involve the computation and storage of the kernel matrix K, which can be memory-intensive and does not scale efficiently with large training datasets.

Multi-Label regression has shown effectiveness in reducing prediction error for the critical pattern P_{min}. The decision to use Regression Chains or multi-label approaches should be guided by the specific objectives. If the focus is on predicting a single label from the available patterns, Multi-Label regression might be superfluous. Conversely, when dealing with highly correlated labels and the goal is to predict all labels, including critical patterns, multi-label regression can yield more accurate models. This approach helps achieve a lower guardband G, thereby enhancing production yield by reducing the number of discarded functional devices.

5.5.5 Active Learning

Three advanced regression algorithms—Ridge regression with polynomial features (PFR), Support Vector Regression (SVR), and Random Forest (RF)—were

Table 5.6: Comparison between advanced models in-chain or single target (P_{min} target), average over 20 run.

Algorithm	Single models			Multi-Label (RC) models		
	nMAE	nRMSE	G	nMAE	nRMSE	G
Ridge	1.15%	1.55%	10.20%	**1.12%**	**1.53%**	**10.00%**
SVR (4 PCs)	1.08%	1.51%	9.99%	1.08%	1.51%	9.99%
ExtraTrees	**1.11%**	**1.51%**	**9.91%**	1.12%	1.52%	9.97%
Lasso	1.15%	1.55%	10.20%	**1.11%**	**1.51%**	**9.92%**
ElasticNet	1.15%	1.56%	10.20%	**1.11%**	**1.51%**	**9.93%**
OMP	1.18%	1.59%	10.41%	**1.15%**	**1.55%**	**10.19%**
SVR (14 PCs)	1.08%	1.50%	9.97%	1.08%	1.50%	**9.95%**
Random Forest	**1.12%**	**1.53%**	**10.04%**	1.13%	1.54%	10.07%
Decision Tree	**1.57%**	**2.15%**	**14.09%**	1.60%	2.18%	14.29%
Ridge RK	**1.24%**	**1.67%**	**10.65%**	1.25%	1.68%	10.72%
Lasso RK	**1.24%**	**1.67%**	**10.67%**	1.25%	1.69%	10.77%
ElasticNet RK	**1.24%**	**1.67%**	**10.67%**	1.25%	1.69%	10.77%
Ridge PK	1.27%	1.70%	10.86%	**1.22%**	**1.66%**	**10.58%**
Lasso PK	1.27%	1.71%	10.88%	**1.22%**	**1.66%**	**10.61%**
ElasticNet PK	1.27%	1.71%	10.88%	**1.22%**	**1.66%**	**10.61%**
Kernel Ridge	1.15%	1.56%	9.90%	**1.12%**	**1.52%**	**9.70%**
MARS	1.33%	1.78%	11.34%	**1.30%**	**1.73%**	**11.05%**
MARS (14 PCs)	1.43%	1.89%	12.07%	**1.40%**	**1.85%**	**11.78%**

compared to assess how these models behave as the quantity of labeled data increases, particularly when samples are selected using active learning (AL) strategies.

Seven AL selection strategies, as described in section 5.4, were employed and compared, alongside a random sampling and labeling approach. This random approach involved training models using a random subset of the available wafers, with the models being tested on an appropriate test set at each step.

The results, presented in table 5.8, indicate that for Polynomial Ridge, good accuracy (below 2% nRMSE) can be achieved with 13 wafers using the random approach, 11 using ED, 8 using QBC and GSx, 7 using GSy and iGS, and only 6 wafers using LOF and HD. Notably, these last two metrics also exhibit the lowest nRMSE variance (although GSx and iGS also show low variance after 9 wafers). Similar results were observed with the other algorithms, as summarized in fig. 5.7, where the advantages of AL techniques over random selection are evident. Initially, the error is high, but it decreases as more wafers are added.

For all the nRMSE curves, the Area Under the Curve (AUC) can be calculated; a lower AUC value indicates a model's ability to generalize well with fewer steps. The random approach yields a relatively high AUC value, but this is not the case when a specific strategy is chosen. Across all the metrics considered, the nRMSE curve declines more rapidly with a strategic approach than with a random one. The nRMSE is halved within a few steps when a precise strategy is employed, and this trend is consistent across all the algorithms.

In the final step, SVR and RF slightly outperform Ridge; however, Ridge demonstrates better generalization in the intermediate steps and has the lowest AUC value under the nRMSE curve. This result suggests that more complex models tend to overfit when the data volume is insufficient.

table 5.7 highlights that LOF and HD are likely the most effective approaches for this problem, yielding the lowest AUC under the nRMSE curve.

Table 5.7: Area under the nRMSE curve for different algorithm (Polynomial Ridge, Random Forest, SVR)

Alg	Rnd	EDD	LOF	QBC	GSx	GSy	iGS	HD
PFR	86.8	57.8	54.2	55.0	54.7	56.1	54.8	**54.1**
RF	93.2	77.8	**62.8**	64.9.0	64.2	66.1	63.1	64.1
SVR	89.3	53.9	52.2	57.0	53.2	55.4	52.8	**52.1**

The error scores obtained at each step of the AL procedure were correlated to the unsupervised metrics and reported in table 5.9. Four metrics were used: mutual information, Pearson, Spearman, and Kendall [125] correlation. All of them show a very strong correlation between the nRMSE and the selection strategies score; also in this case, LOF and HD seem to be the best metrics to be used for the AL. A graphical view of the correlation can be observed in fig. 5.8 where mean values are reported.

5.5.6 Discussion on Active Learning

The impact of the AL procedure can be quantified by measuring the time required to create the ML training set. Consider the scenario where no labeled devices are available at the outset. Acquiring labels for each device requires at least 30 minutes. Each wafer typically consists of 200 devices. Using a random sampling and labeling approach, achieving an acceptable error (below 2% nRMSE) necessitates 13 wafers, whereas only 6 wafers are needed with AL (LOF strategy), as indicated in fig. 5.7. This implies that, with a random approach, dataset creation would take at least $30\text{min} \times 13\text{wafers} \times 200\text{devices} = 78{,}000\text{min} \approx 55\text{days}$. By

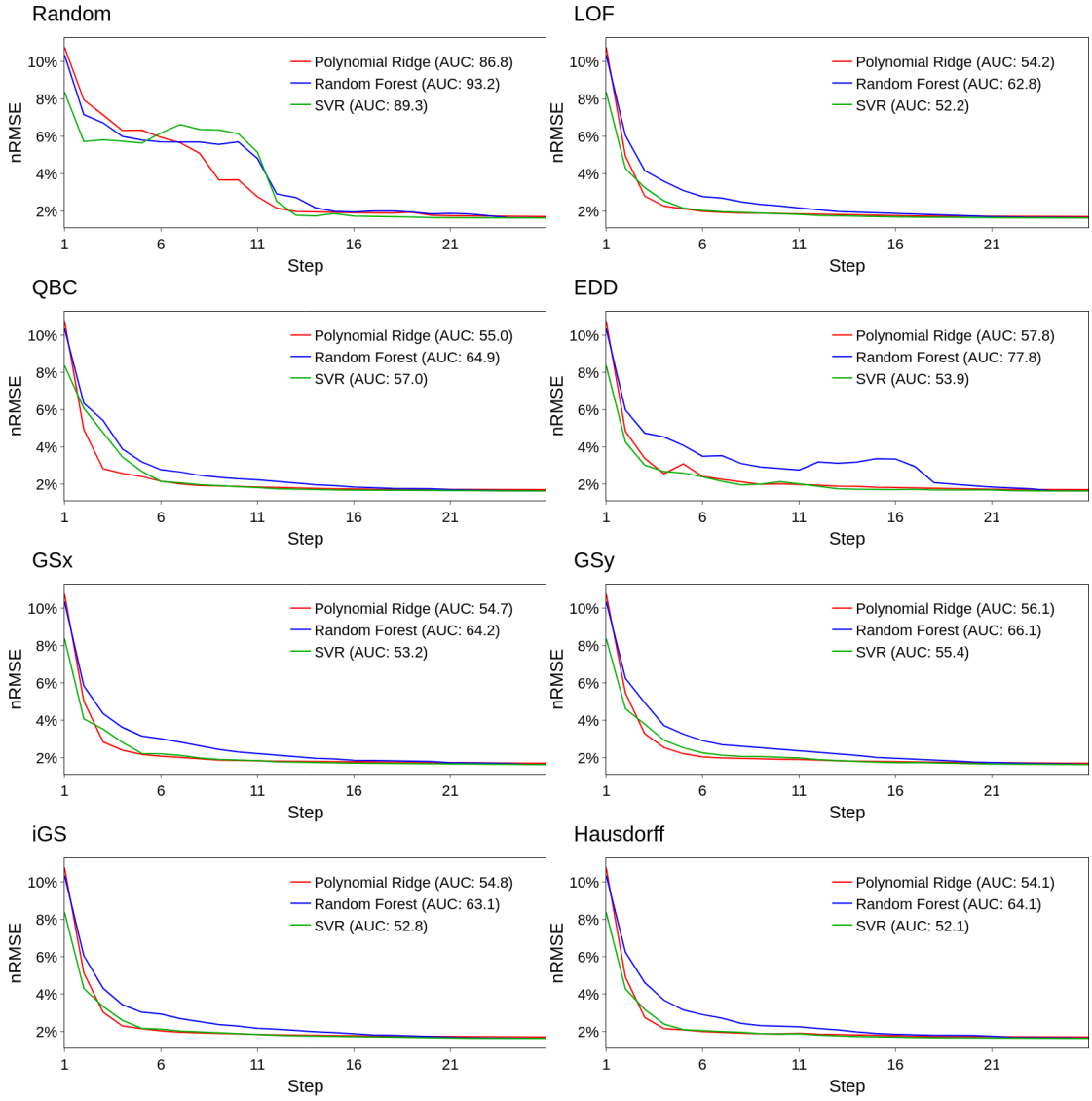


Figure 5.7: nRMSE obtained with different regression algorithms, using AL strategies. The curves represent the mean value of the errors depending on all possible initial wafer choices.

contrast, employing AL reduces the dataset creation time to less than half ($\approx$ 25 days).

In practice, it is not feasible to test model performance at each step of the AL procedure (as done in fig. 5.7) due to the unavailability of a test set. This limitation applies to both random sampling and labeling. However, when using AL, an estimate of model performance can be obtained by examining the values of the selection strategies on the unlabeled data, as these are highly correlated

with prediction error (fig. 5.8, table 5.8). Low values for AL strategies generally correlate with low prediction errors. The concept of a 'low value' depends on the strategy used and is dataset-dependent. Establishing thresholds requires sufficient data to build a proper test set.

In any case, models created using the AL procedure are more likely to be of higher quality at each step compared to those generated by random sampling and labeling. AL metrics can also be applied to wafers from production lines. If a particular wafer exhibits an AL metric value exceeding a certain threshold (computed based on values observed during the training set building), it can be labeled and included in the training set, allowing the model to be updated. This approach helps prevent the ML model from becoming obsolete, which may occur due to various reasons, such as shifts in production processes. Thus, AL can be viewed as a method for performing novelty detection in production, identifying potential causes of process variation and dataset shift.

Table 5.8: Polynomial Ridge Regression Mean and variance nRMSE during active learning using all possible initial wafer choices

Step	nRMSE mean [%]								nRMSE variance [%]							
	Rnd	ED	LOF	QBC	GSx	GSy	iGS	HD	Rnd	ED	LOF	QBC	GSx	GSy	iGS	HD
1	10.76	10.76	10.76	10.76	10.35	10.35	10.35	10.35	3.29	3.29	3.29	3.29	4.02	4.02	4.02	4.02
2	7.95	4.84	4.95	4.92	5.83	6.25	6.05	6.25	2.96	2.68	2.71	2.54	1.23	1.63	1.61	1.36
3	7.15	3.39	2.80	2.83	4.36	4.93	4.31	4.61	2.98	0.70	0.57	0.63	1.17	1.65	0.85	0.99
4	6.31	2.56	2.26	2.58	3.62	3.71	3.43	3.68	2.62	0.54	0.23	0.46	0.54	0.90	0.60	0.90
5	6.33	3.09	2.12	2.41	3.16	3.26	3.03	3.15	2.58	0.97	0.16	0.53	0.26	0.74	0.26	0.31
6	5.95	2.41	1.99	2.15	3.01	2.91	2.93	2.90	2.45	0.47	0.16	0.21	0.28	0.31	0.31	0.22
7	5.65	2.26	1.93	2.02	2.83	2.70	2.69	2.71	2.25	0.28	0.09	0.13	0.31	0.27	0.22	0.24
8	5.09	2.11	1.89	1.92	2.63	2.62	2.52	2.43	1.85	0.27	0.06	0.08	0.29	0.41	0.22	0.14
9	3.66	1.99	1.89	1.91	2.44	2.53	2.37	2.32	1.26	0.16	0.07	0.09	0.16	0.44	0.17	0.11
10	3.68	2.01	1.87	1.87	2.30	2.44	2.28	2.27	1.31	0.11	0.06	0.08	0.19	0.35	0.17	0.12
11	2.77	1.99	1.83	1.84	2.22	2.36	2.17	2.25	0.48	0.09	0.05	0.06	0.13	0.32	0.15	0.13
12	2.16	1.94	1.82	1.82	2.15	2.27	2.12	2.16	0.11	0.07	0.06	0.06	0.16	0.28	0.15	0.12
13	1.98	1.89	1.80	1.78	2.07	2.20	2.05	2.09	0.06	0.05	0.05	0.03	0.13	0.21	0.16	0.09
...																
26	1.70	1.70	1.70	1.70	1.70	1.70	1.70	1.70	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

5.6 Optimize the training set

As discussed in section 5.1, the baseline approach for managing outliers and missing values involves removing them from the training set. However, as shown in section 5.5, this method results in the elimination of approximately 50% of the available samples. Given that labeling each sample requires significant time, this approach can lead to a substantial waste of resources.

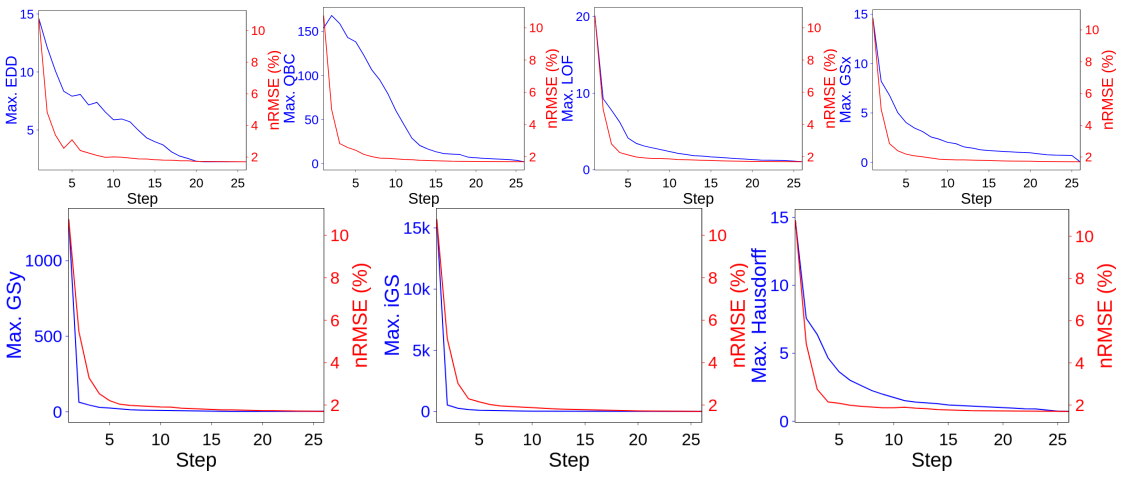


Figure 5.8: Correlation between active learning unsupervised scores (EDD, LOF, QBC, GSx, GSy, iGS, HD) and nRMSE using polynomial Ridge regression.

Table 5.9: Correlation between selection strategies and nRMSE on the $f_{\max}$ Ridge regression models with polynomial features

Selection strategy	Mutual information	Pearson correlation	Spearman correlation	Kendall correlation
EDD	0.928	0.708	0.927	0.782
LOF	0.889	0.828	0.967	0.849
QBC	0.898	0.526	0.973	0.854
GSx	0.802	0.839	0.967	0.857
GSy	0.944	0.643	0.822	0.609
iGS	0.935	0.611	0.953	0.817
HD	0.859	0.843	0.968	0.857

As outlined in sections 2.5 and 5.5.3, the patterns are highly correlated, and the dataset contains multiple labels. Owing to this high degree of correlation, missing label information can be inferred from other measurements. Thus, it is possible to preprocess the noisy dataset by attempting to: (i) identify anomalous and noisy data within the portion of the dataset used for training, and (ii) recover samples that lack label measurements, rather than discarding them. This optimized dataset enables earlier termination of the characterization phase while achieving the same amount of labeled samples and improves the quality of the training data.

The procedure comprises three steps, which will be elucidated using a simplified dataset and graphically represented in fig. 5.9, excluding features. For each of the six samples in the toy dataset, six distinct labels are collected through multiple measurements per label. The frequency values depicted in the figure are normalized

based on the mean of the corresponding label. In this example, we will focus on labels, though the approach can be extended to features.

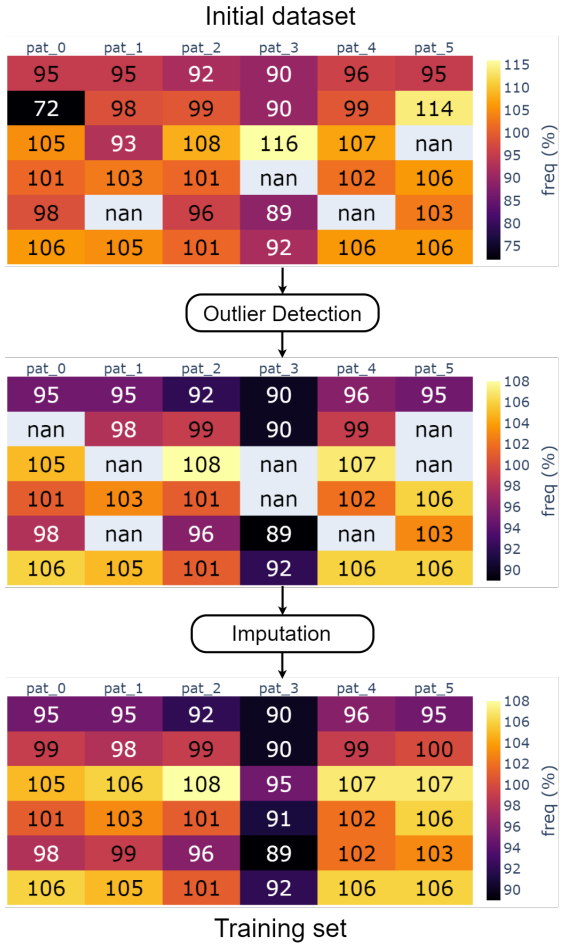


Figure 5.9: Steps performed during the data preparation phase on the labels dataset.

In the first step (upper dataset in fig. 5.9), noisy data are identified by examining the variance of the multiple measurements of each label (see fig. 5.2). Labels exhibiting high variance are discarded and replaced with a not-a-number (NaN) value, thereby introducing missing values into the dataset. This initial step serves as a "filter" applied by domain experts, with the threshold value contingent upon the quality of the measurement process. Outliers may evade this procedure, necessitating the use of supplementary detection methods.

The second step involves the application of statistical methods for outlier detection, specifically based on the interquartile range (IQR) and z-score (as depicted

in section 3.8), to identify label measurements that deviate from normal distributions (see fig. 5.1). The analysis is conducted per label and can also be performed per wafer (i.e., when computing IQR or z-score, only samples within the same wafer are considered). In the upper dataset of fig. 5.9, the second sample contains two outlier labels (the first and the last patterns), which are replaced with NaN values in the middle dataset. If the majority of labels for a given sample are NaN, that sample should be discarded.

In the final step, imputation is employed to fill in the missing values. This approach allows for the retention of samples and the recovery of information from incomplete rows. However, simple univariate imputation, which replaces the missing values of a feature with the mean or median of that feature in the dataset, may not be appropriate. This method is overly simplistic and can introduce additional noise into the data. Given the high correlation among labels in the problem at hand, it is reasonable to construct a model, even a simple linear one, capable of predicting the value of a target label based on the others. This rationale supports the use of a multivariate imputation model, which predicts missing values by considering multiple columns of the dataset. The resulting values are shown in the lower dataset of fig. 5.9. Here, we treat both missing and removed data similarly, since devices with high-variance repeated measurements are preemptively excluded from the dataset (as detailed in section 5.1).

The final choice is to combine a z-score and IQR in a double-check fashion, marking as NaN those labels that exhibit a z-score higher than 3 or that fall outside the IQR range. These procedures can be conducted at the whole-dataset level or on a per-wafer basis. The wafer-level approach accounts for process variations among corner-lots and the presence of faster and slower wafers.

Furthermore, to remove samples that may be unreliable, devices with a number of missing labels exceeding a certain *threshold* (equal to 5) are discarded, while the remaining devices are imputed. This approach utilizes 87% or more of the available samples. This final approach is grounded in the notion that the imputation procedure is deemed unreliable when the number of available labels is insufficient, even if the correlation among the labels is nearly perfect.

5.6.1 Experiments

For the imputation procedure, the Iterative Imputer from scikit-learn [105] was employed, using Random Forest as the base estimator to replicate the MissForest imputation algorithm [131]. It was observed that a simple linear regression performs comparably well in this context, due to the extremely high linear correlation among the labels (table 5.5) The regression algorithms utilized include: 1) Ridge Regression, which involves a preprocessing pipeline with input normalization, PCA

feature reduction to 14 components, and polynomial transformation, and 2) Kernel Ridge Regression, which incorporates input normalization and the radial basis function (RBF) kernel. Both algorithms are implemented in a multi-output manner using the Regression Chain approach developed [129] in section 5.5.3.

Table 5.10 summarizes the performances of machine learning algorithm when trained on different training sets, each derived using a unique outlier detection and imputation schemes. The first three columns of the table outline the methods employed in deriving the training sets. Column 1 (*Missing values*) specifies whether missing values were handled by *imputation* or by *dropping* the corresponding samples. Column 2 (*Baseline outliers*) indicates whether anomalous data, identified by looking at the *Outlier Index* values, were *kept*, replaced with *NaN* followed by *imputation*, or if the affected samples were *dropped*. Column 3 (*Outlier detection*) details whether additional outliers on labels were identified at the *dataset-level* or *wafer-level*, or if no further detection (*None*) was applied. Column 4 (*Training set size*) displays the number of samples used in the training process. Finally, the last two columns present the normalized root mean squared error (*nRMSE*) and the mean absolute error (*nMAE*) of the model on the test set.

Table 5.10: Prediction performances of $P_{\min}$ target on test set with different methods for outlier detection on labels (Polynomial Ridge Regression)

	Missing values	Baseline outliers	Outlier detection	Training set size	nRMSE	nMAE
0	Drop	Keep	None	2,058	1.67%	1.27%
1*	*Drop*	*Drop*	*None*	*1,805*	*1.59%*	*1.19%*
2	Imputation	Keep	None	3,437	1.66%	1.28%
3	Imputation	Keep	Dataset-level	3,437	1.66%	1.28%
4	Imputation	Keep	Wafer-level	3,437	1.58%	1.20%
5	Imputation	Imputation	None	3,437	1.58%	1.20%
6	Imputation	Imputation	Dataset-level	3,437	1.57%	1.19%
7	Imputation	Imputation	Wafer-level	3,437	1.59%	1.21%
8	Imputation	Drop	None	3,014	1.57%	1.18%
9	Imputation	Drop	Dataset-level	3,014	1.56%	1.16%
10	Imputation	Drop	Wafer-level	3,014	1.57%	1.18%
11	Threshold	Imputation	Wafer-level	3,343	1.60%	1.22%
12	Threshold	Drop	Wafer-level	2,986	1.56%	1.17%

* Baseline training set and model

The first two models presented in table 5.10 deal with the removal of baseline outliers (the ones with *Outlier Index* > 0). In particular, the second model represents the baseline, illustrating how the removal of baseline outliers enhances model performance. The subsequent three models (2-3-4) demonstrate the combination of imputation with outlier detection, allowing for the utilization of 100% of the available samples in the training process. The next two groups (models 5-6-7 and 8-9-10) indicate that baseline outliers should either undergo imputation or be discarded to surpass the baseline model's performance; these models utilize approximately 88% of the available samples. In the final group, a mixed strategy is employed, where devices exhibiting more than a specified threshold (five, in this case) of missing labels are discarded, while the others are imputed. This approach further improves performance, with these models using at least 87% of the available samples. The rationale behind this final approach is the unreliability of the imputation procedure when the number of available labels is low, despite the near-perfect correlation among them.

Comparable results can be achieved with both linear and kernel-based algorithms, yielding an nRMSE of 1.56% and 1.50%, respectively. The coefficient of determination (R^2) is 0.9842 for the baseline model (2) and 0.9846 for the best-performing approach (12). The increased sample size contributes to the development of more robust models by exploring a larger portion of the input space.

It is noteworthy that models 5-7 and 8-10 yield similar results: when imputation is applied, the method of handling baseline outliers (whether dropping or imputing them) has minimal impact on nRMSE and nMAE, which remain close across these models. In this scenario, the imputation process appears to add little valuable information, as the imputed values likely fall within the distribution's tails, rendering them less informative. Including these samples in the training set could potentially distort the training process.

The most promising strategies were compared with the baseline model using learning curves for both regression algorithms, as illustrated in fig. 5.10. The x-axis represents the number of characterized devices available in the optimal case (i.e., when 100% of the devices are used for training); the percentage of actual samples in the training set relative to the available devices depends on the strategy employed and reflects the results presented in table 5.10. The y-axis displays the nRMSE of the model on the test set.

All models were evaluated on the same test set during each step of the learning curve creation.

The learning curves demonstrate the potential for reducing the number of devices that need to be characterized in order to achieve a desired level of accuracy. In fig. 5.10, focusing on Ridge Regression (upper figure), the yellow curve (corresponding to model 12 in table 5.10) attains the final accuracy of the baseline method (i.e.,

1.59% nRMSE, represented by the red curve) after only 1,000 samples, and consistently remains below this threshold. This indicates that characterizing just 1,000 samples—less than 30% of the devices required for the baseline approach—could suffice for training. Similar trends are observed with Kernel Ridge Regression (lower figure). Consequently, a significant reduction in the human effort required for label measurement can be achieved.

In summary, the imputation technique offers two key advantages: 1) minimizing the number of samples that are dropped, thereby increasing both the training set size and model performance, and 2) reducing the number of devices required for characterization to reach satisfactory accuracy, leveraging the available labels to reduce the time and cost associated with measurements.

5.7 Proposed Overall Flow

Following the description of data preprocessing, the development of ML predictive models, and the AL strategies for wafer selection in the training set, an overview of the complete workflow is provided, as depicted in fig. 5.11.

The process begins with the collection of devices from production lines, grouped by their respective production wafers (corner-lots), each containing several hundred devices. Each wafer is represented as a point in the dataset, with its *centroid* calculated by averaging all features of the devices it contains. The methodology requires at least one labeled wafer to initiate; if no labeled data is available, an initial selection strategy must be employed, as outlined in [151]. Subsequent steps proceed identically, irrespective of how the initial wafer is selected.

For wafers without labels, an AL metric is computed to assess their *distance* from the existing training set. The wafer exhibiting the highest utility score is labeled and incorporated into the training set. This iterative process continues until a predefined stopping criterion is reached. The labeled wafers constitute the training dataset for subsequent ML model development, which can utilize multi-label regression techniques, such as Regression Chain. During the testing phase, the SMON values of new wafers are collected to predict the maximum operating frequency. Devices predicted to have frequencies below a certain threshold are classified as defective and removed from production.

Additionally, AL metrics may be applied to wafers from production lines. If a wafer's score surpasses a predetermined threshold—set during the initial wafer selection phase—it can be labeled, added to the training set, and used to refine the ML model. This methodology facilitates:

- The identification and inclusion of the most informative samples, optimizing the training set size and reducing characterization time.

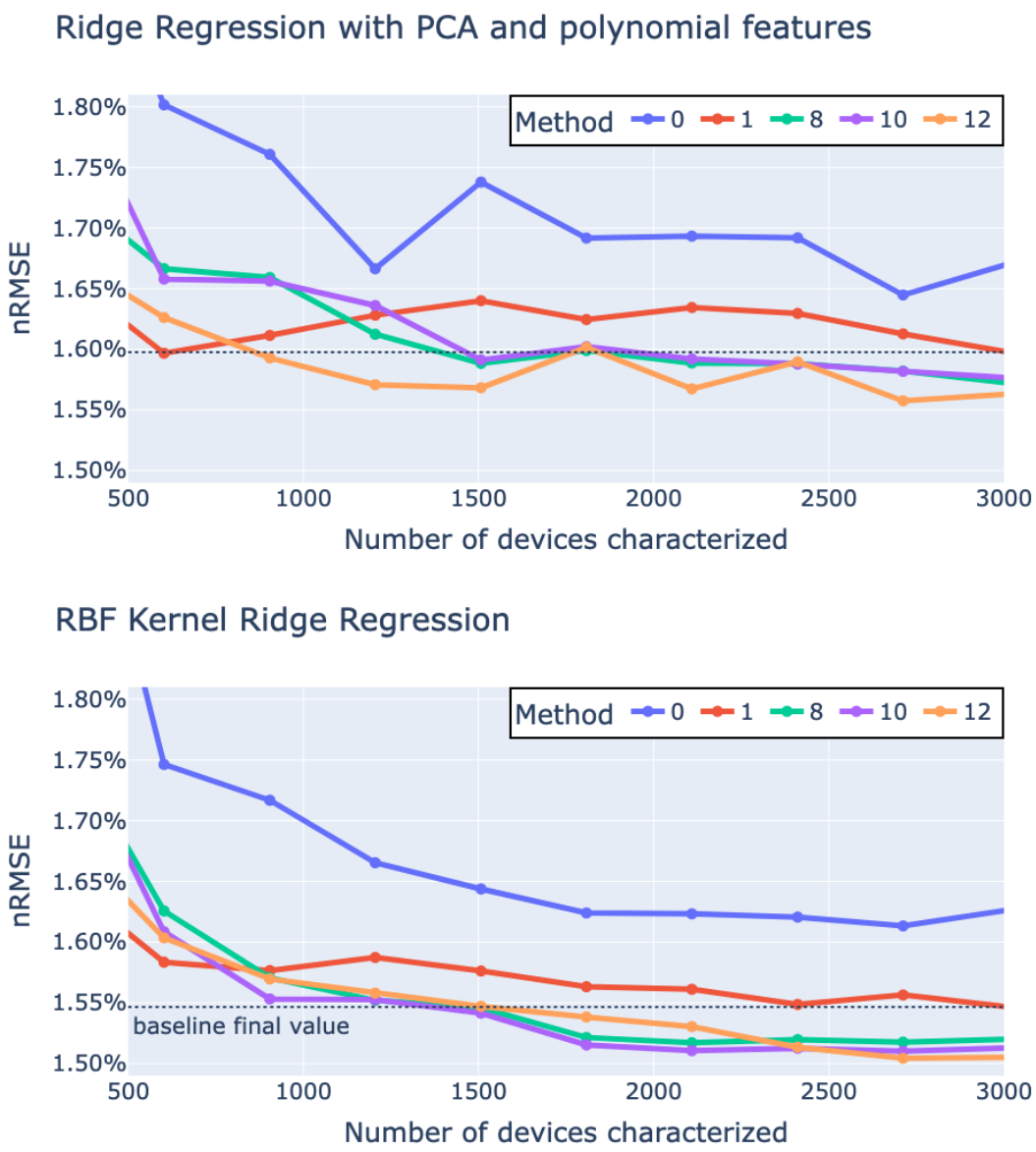


Figure 5.10: Learning curves of Ridge (upper plot) and Kernel Ridge (lower plot) models with different types of training sets, tested on the same test set. The x-axis represents the number of devices characterized (constant), while the y-axis indicates the nRMSE. The dashed lines represent the nRMSE values achieved by the baseline models (1, red) with the maximum number of samples in the training sets.

- The development of robust machine learning models that exploit label correlations to enhance both predictive accuracy and production yield.
- The continuous updating of the model with critical wafers from ongoing production, ensuring sustained model relevance and precision.

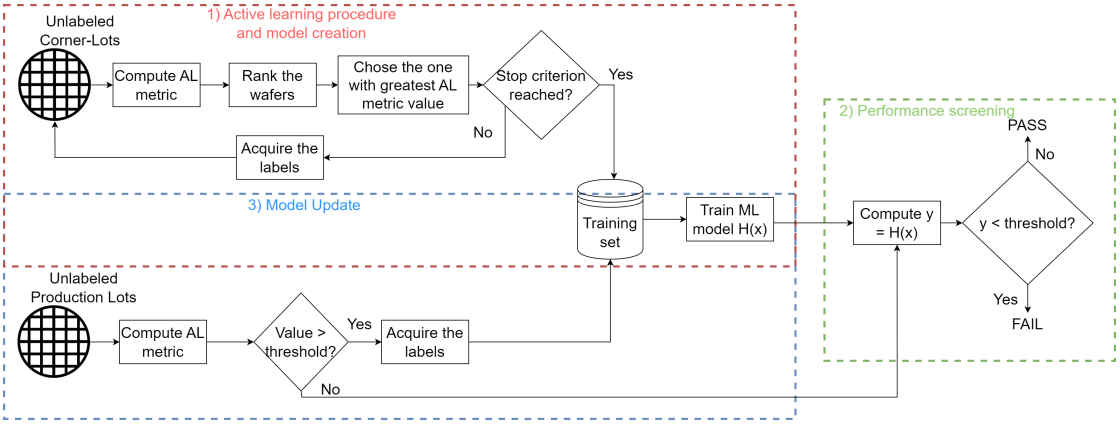


Figure 5.11: Proposed performance screening flow

5.8 Conclusion

This chapter laid the foundation for an ML-based framework designed to predict the maximum operating frequency of MCUs. The framework leveraged well-established ML techniques for performance screening during production, incorporating AL and multi-label regression.

The contributions of this study were twofold: (1) the enhancement of existing performance prediction methods through feature-transformation approaches and multi-label learning, which resulted in lower prediction errors and reduced guard-bands, thereby increasing process yield; and (2) the development of a method to minimize the number of labeled devices required to construct ML models, significantly reducing the time needed for dataset creation.

Performance prediction models, as discussed in [44], have been enhanced using dimensionality-reduction techniques, which effectively decrease the number of features analyzed by the ML models. The introduction of polynomial transformation enables the creation of more suitable non-linear models, further reducing prediction errors when compared to baseline methods. These approaches facilitate the use of simpler, more interpretable linear models.

The multi-label learning approach has proven effective in improving regression performance by exploiting relationships and correlations among target variables.

A significant reduction in the prediction error for the critical pattern $P_{\min}$ was achieved. While most regression algorithms benefit from the Regression Chain technique, tree-based estimators did not show improved performance, likely due to overfitting issues.

AL techniques have demonstrated considerable effectiveness in reducing the number of labeled points required to reach a desired error threshold by optimizing the training set. The LOF and HD strategies emerged as the most effective selection methods for production, capable of halving the training set size compared to random sampling, which results in significant time savings—from 55 days with random sampling to 25 days with the LOF/HD AL approach, maintaining prediction accuracy. Additionally, AL can provide an indication of a model's generalization error by assessing the distance of test samples from the training set. Samples that are farther from the training distribution often highlight areas where the model is less confident, offering insights into potential weaknesses and guiding further training efforts.

In conclusion, the combined use of AL and multi-label regression enables the faster and more effective construction of robust ML models, achieving lower prediction errors, reducing guardbands, and increasing yield. The methods presented can be applied to any scenario where the objective is to correlate alternative measurements with device performance, utilizing ML algorithms to establish relationships between features and labels. Multi-label techniques are particularly advantageous in scenarios involving multiple highly correlated labels, whether predicting each individually or deriving a final target from them, as is the case with $P_{\min}$.

Techniques for addressing anomalous and noisy values in labels within the context of MCU performance screening have been presented. The detection of outliers is a critical step, as they significantly impact the performance of ML models. Results indicate that imputation is a suitable approach in this context, as it increases the number of available samples for the training process, thereby improving dataset quality. This approach effectively enhances the performance of existing methods for MCU performance screening. Specifically, the dual objectives achieved include: (1) the creation of a higher-quality dataset, and (2) the reduction in the number of devices requiring characterization by a factor of three compared to the baseline method. This methodology is applicable to other domains characterized by measurement errors, noise, and highly correlated labels.

Chapter 6

Deep Learning in MCU Performance Screening

In the previous chapter, an ML framework was developed for MCU performance screening. However, several critical challenges remain unaddressed. A significant issue impacting the accuracy of supervised ML models is the dependence on the availability of high-quality labeled data. Unfortunately, the process of collecting $F_{\max}$ data is labor-intensive and challenging, as discussed in sections 2.5 and 5.1. Developing a new ML model from scratch under such conditions is time-consuming, with dataset acquisition potentially requiring several months.

This challenge becomes particularly prominent when introducing new MCU products. Utilizing pre-existing models trained on previous-generation data may lead to inaccurate predictions due to shifts in data distributions, necessitating retraining or fine-tuning. Nevertheless, the fine-tuning of shallow learning models, such as those outlined in the earlier chapters, remains impractical due to the intrinsic characteristics of their learning algorithms. These models, which encompass traditional ML algorithms such as SVMs, Decision Trees, and k-Nearest Neighbors, typically lack optimization methods that enable an online learning framework. Conversely, this capability is more readily attainable in complex models, such as those employed in deep learning (DL).

One of the critical distinctions between shallow and deep learning models lies in the optimization process. Deep learning models often utilize gradient descent-based methods to minimize a loss function by iteratively adjusting the model parameters. Gradient descent, including its variants such as Stochastic Gradient Descent (SGD) and Adam, leverages the model's continuous and differentiable nature to navigate complex loss landscapes. This process enables deep learning models to fine-tune their parameters over multiple layers, capturing intricate patterns in large datasets.

In contrast, shallow models generally do not operate within such a continuous, multi-layered parameter space. Their training procedures often involve non-differentiable operations, discrete optimization, or even greedy algorithms that do not naturally lend themselves to gradient-based optimization. For instance, decision trees are constructed through recursive partitioning of the feature space, where each split is selected based on a local criterion, rather than a global optimization objective. Similarly, SVMs optimize a convex objective function with constraints, where the solution is obtained by solving a quadratic programming problem, not through gradient descent.

Moreover, the parameter space of shallow models is typically less flexible, with fewer degrees of freedom. This rigidity can lead to challenges when attempting to fine-tune these models post-training, as the scope for parameter adjustment is limited and does not easily allow for iterative refinement in the manner that deep models do.

Consequently, the fine-tuning strategies that are effective for deep learning models are not directly transferable to shallow learning models. The distinct nature of their optimization processes, coupled with their different structural characteristics, underscores the need for alternative approaches when optimizing shallow models, which may involve heuristic methods, cross-validation for hyperparameter selection, or ensemble techniques, rather than fine-tuning in the traditional sense.

Thereby, adhering to shallow learning models necessitates the training of new models from scratch rather than relying on fine-tuning existing models.

This chapter aims to address these challenges by optimizing the ML model training process to reduce reliance on extensive labeled datasets, minimize prediction errors, and establish a framework for constructing predictive models that are reusable across different product families.

To achieve these objectives, DL techniques are employed to extract relevant features and patterns from unlabeled data collected from production lines, using semi-supervised learning (SSL) and Transfer Learning approaches. DL models are utilized as feature transformers, with the transformed data subsequently used to train a simple linear regression model for predicting $F_{\max}$. The DL models initially developed for one MCU product (*A1*, as referenced in chapter 5) are further employed in a Transfer Learning context to construct predictive models for a different product (*A2*). This approach accelerates the deployment of ML models in production by requiring only shallow linear regression models trained on features extracted by the DL models. Additionally, Transfer Learning techniques from a legacy product family (*A*) are leveraged to address data scarcity when releasing a new MCU product (*B1*) from a different family (*B*), thereby significantly reducing the effort and time required for label acquisition. Leveraging insights and models from previous products significantly reduces the time required in ongoing research

and development processes.

The proposed framework establishes a robust foundation for further exploration. Testing and ML engineers can utilize this framework as a reliable benchmark when developing novel models and techniques to enhance performance prediction accuracy, particularly when insights into data, SMONs interactions, and the required number of labeled samples for training reliable models are incomplete.

By leveraging a diverse set of MCU samples (products $A1$, $A2$, B), the framework aims to capture inherent characteristics common to different products within the MCU domain. This approach not only enhances the generalizability of the findings but also addresses a gap in the existing literature.

6.1 Motivation

In section 2.5, the challenges of obtaining labeled data for MCU performance screening were discussed. In scenarios where data scarcity is a concern, techniques such as Active Learning (section 5.5.5) and Outlier Detection (section 5.6.1) have proven effective in efficiently creating an informative training set, thereby facilitating the development of robust ML models. AL techniques enable the selection of the most informative samples, resulting in a higher quality training set. Additionally, anomalies in the labels introduced during the measurement phase can adversely affect ML regression models in MCU performance screening, as demonstrated in section 5.6.1, where a method to recognize and replace anomalous labels in the training set was presented.

However, while these approaches are effective and can halve the size of the training set, they may still be insufficient in contexts with a very limited number of samples. Even with a high-quality dataset, the limited sample size may result in biased models that struggle to generalize to unseen data.

The initial approach to building a new ML model from scratch involves starting with the early phase of obtaining labeled data and training a shallow learning model. This process, however, requires waiting until a sufficient amount of labeled data is collected to train a reliable model across the feature domain of interest and evaluate its performance on an appropriate test set. This phase is time-consuming: considering that labeling a single sample takes at least 30 minutes (section 5.6.1, [44]), obtaining a suitable training set of approximately 3000 samples, with no missing values or outliers, would require 63 days of continuous labeling. If noise in the labeling process is not considered and samples with missing values and outliers are dropped (one in two, as indicated in section 5.1), this time would double. Even with measures to handle missing values (as outlined in section 5.6.1), the possibility of outliers and the consequent dropping of samples (about one in four, section 5.6.1) must be considered, increasing the time by a factor of 1.25. This duration does not

account for the feature engineering phase, which is often the most time-consuming aspect of developing a predictive model and may take additional months.

In contexts with scarce data, traditional shallow learning models (e.g., SVM, Ridge Regression, ElasticNet) typically outperform DL models due to the increased complexity of the latter, which often leads to overfitting. However, shallow learning requires precise feature engineering as a preprocessing step to improve model performance. This involves incorporating domain knowledge, such as applying polynomial transformations to raw features before feeding them into a Linear Regression model (chapter 5). In contrast, DL models do not necessarily require this step or do not consider it as crucial.

DL models have garnered significant attention due to their superior accuracy when applied to large datasets. These models are capable of learning hierarchical data representations through nested concepts, with each concept defined in relation to simpler ones [66]. They autonomously extract features and identify the best data representation by manipulating input features, rather than relying on a priori handcrafted features, as is common in shallow learning. Additionally, the architecture of DL networks can be tuned with great flexibility.

However, in cases where only limited labeled data are available, the aforementioned DL models may not be applicable. This situation commonly arises during the early stages of manufacturing a new product when only a few data samples are available. Nevertheless, DL models are flexible and can be pre-trained on a large training set (often comprising millions of samples) and then fine-tuned on a related dataset to solve a similar task. This approach is the foundation of Transfer Learning (see section 3.6.7). Instead of waiting to collect a sufficient number of labeled data, SSL and Transfer Learning can be employed. Specifically, Transfer Learning enables the use of only a portion of the architecture of a DL model (e.g., the first n layers) to extract relevant feature embeddings, with the models serving as feature transformers or encoders.

When an MCU product reaches large-scale mass production, millions of unlabeled data points are generated, which can be used to enhance the knowledge gained from subsequent data analysis. Additionally, when a new product is engineered and deployed, data and models from previous products are often still available. Although applying the exact model trained on legacy products to new ones is infeasible due to significant prediction errors, the models from previous products contain valuable information that can be leveraged to develop new models for the new product. In particular, the latent space discovered by deep neural networks can remain useful, making these models suitable as feature extractors. Only the final layer, responsible for linking the extracted features to the target variable, needs to be trained. If the features extracted by deep neural networks are sufficiently general, only a few samples would be required to train these final

supervised models.

For these reasons, DL models become a natural choice when transitioning to SSL and Transfer Learning in situations with limited labeled data. The rationale behind using Transfer Learning becomes evident: save time during the data collection phase by leveraging knowledge acquired from similar, albeit not identical, domains. This is achieved by transforming the input features using a pre-trained network and training a shallow supervised linear model to link these features to the target variable. This approach enables the development of models that are transferable across different MCU products, providing a foundation applicable in various contexts. These models establish a robust baseline for further investigation and refinement.

6.2 New Products

This section provides an overview of the various products and datasets under consideration. As discussed in the preceding section (section 6.1), when new products are engineered and released, it is necessary to develop new $F_{\max}$ predictive models. However, each product may exhibit distinct characteristics. The quantity and configuration of SMONs integrated into a MCU vary significantly depending on the product type, with numbers ranging from a few tens to several hundreds. Within a legacy product technology family, denoted as A and analyzed in the previous chapters, the count of SMONs remains consistent across different products within the family (e.g., $A1$, $A2$, etc.). However, in a distinct and newer product technology family, labeled as B, the number of SMONs, as well as their design, can differ considerably from one product to another (e.g., $B1$, $B2$, etc.).

In the legacy product family A (section 5.5), each die is equipped with 27 SMONs, reflecting a stable configuration across the family. By contrast, in the new product family B, the MCUs are characterized by a markedly higher number of SMONs, approximately 150, distributed across the die. This substantial increase in the number of SMONs introduces additional complexity and variability in the design and performance evaluation of the products within family B. The expanded SMON configuration in family B necessitates advanced strategies for data handling and analysis to effectively leverage the increased monitoring capabilities offered by these additional sensors.

The implications of this variation are profound. In family A, the uniformity of SMONs simplifies the application of ML techniques, allowing for consistent data acquisition and model development across the product line. Conversely, the heterogeneity in SMONs within family B poses challenges for direct model transferability and demands more sophisticated approaches to feature extraction and dimensionality reduction.

The integration of a larger number of SMONs in family B also offers potential advantages, as it enables more granular monitoring of the MCU's operational parameters. However, this benefit comes at the cost of increased complexity in the data analysis process. Effective utilization of the available SMON data requires careful consideration of the trade-offs between model complexity and generalization capability, particularly when applying ML techniques across different products within family B. The variability in SMON configuration underscores the necessity for adaptive ML models that can accommodate the diverse monitoring environments presented by products in different technology families.

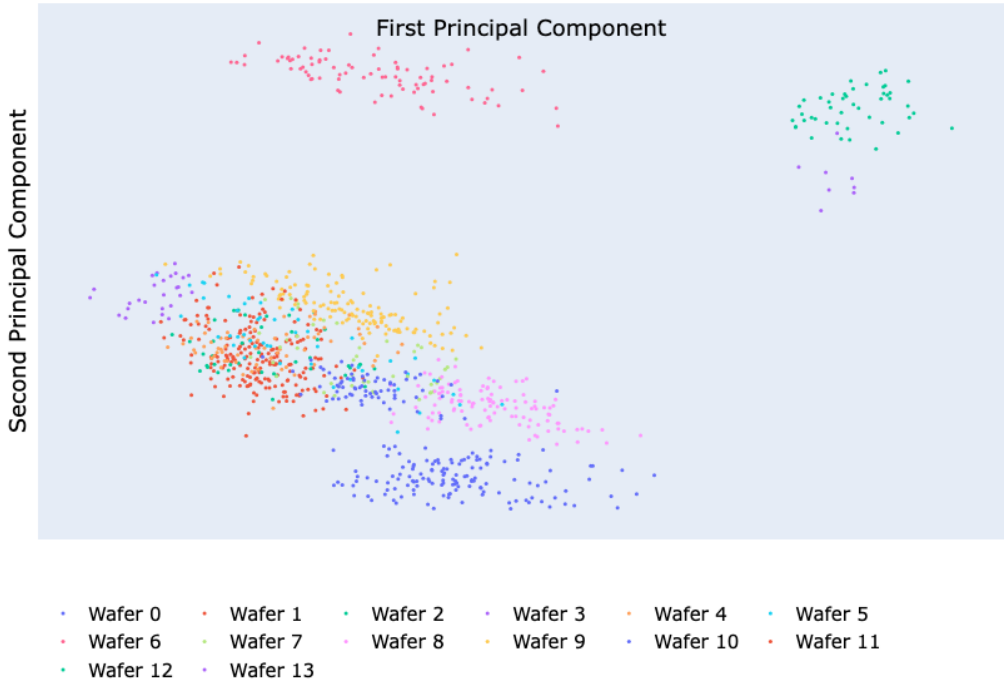


Figure 6.1: PCA-based two-dimensional representation of the corner-lot devices used for the initial model's characterization, colored by wafer, product $A2$

6.3 Proposed Approach

The methodology adopted focus on the extraction of a feature representation from the input data, specifically the SMON values, utilizing DL models functioning as transformers. Semi-Supervised Techniques are employed to pre-train the DL models on both unlabeled data from production lots, comprising millions of samples, and labeled data from corner lots. The objective is to learn a latent space in which projecting SMON frequency values, thereby enabling the DL models to

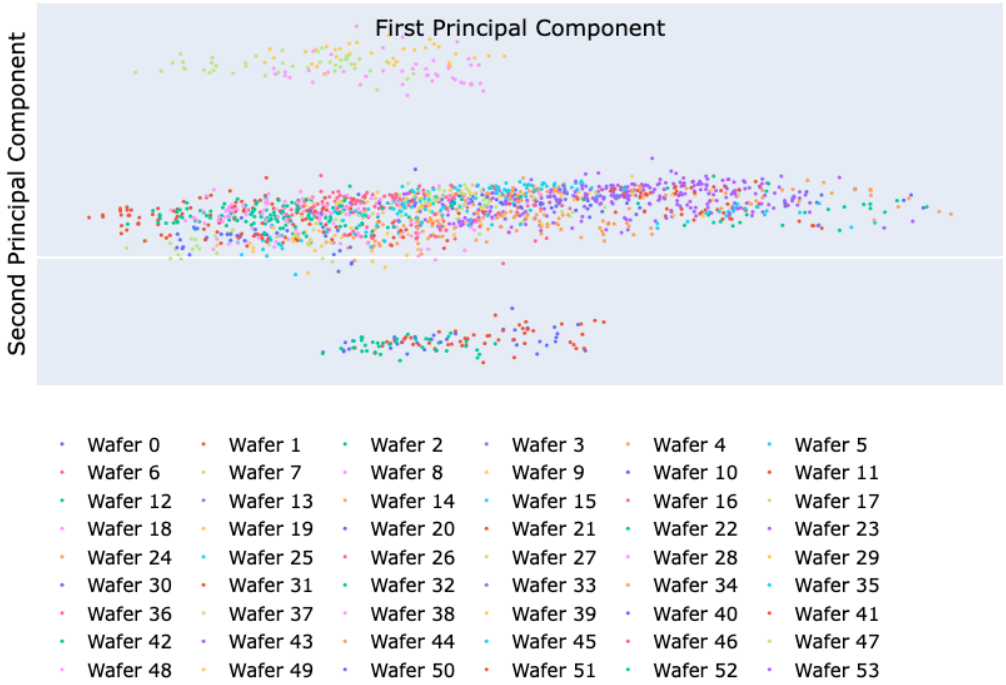


Figure 6.2: PCA-based two-dimensional representation of the corner-lot devices used for the initial model's characterization, colored by wafer, product B. Here the distinction among wafer is less marked.

compute non-linear feature interactions. With the transformed SMON values, the subsequent supervised task, which involves linking the $F_{\max}$ label to the transformed SMON values, can potentially be addressed using a linear model, such as Ridge Regression.

The primary objectives of this approach are as follows:

- Enhance the predictive capability of the models compared to traditional shallow learning approaches and manual feature engineering.
- Minimize the volume of labeled data required to train these models, thereby reducing the time and effort involved in the dataset creation phase, particularly in labeling samples.

Following the pre-training of deep feature extractors on millions of unlabeled samples from a particular product, $A1$, these extractors can be applied to different products with an identical SMON set (referred to as $A2$) or even to products with a completely different SMON set (referred to as $B1$). Given the similarity of feature domains across different products, the aim is to transfer the knowledge—i.e.,

the feature transformation—acquired by the DL models trained on production unlabeled data of a certain product to the labeled data of either the same or different products. This process is designed to minimize the use of labeled data and the necessity for extensive modifications to the deep model.

The research thus seeks to implement a transfer learning framework encompassing two types:

- **Intra-Family:** The deep learning models are trained on product $A1$ from product family A, and the same model is then utilized as a feature extractor for another product, $A2$, within the same family A. Products within the same family share the same SMON set.
- **Inter-Families:** The deep learning models are trained on product $A1$ from product family A, and the same model is subsequently employed as a feature extractor for a different product, $B1$, from a different family B. Products from different families possess different SMON sets, necessitating appropriate adjustments to accommodate the differing input spaces.

The adopted Transfer Learning framework is illustrated in fig. 6.3.

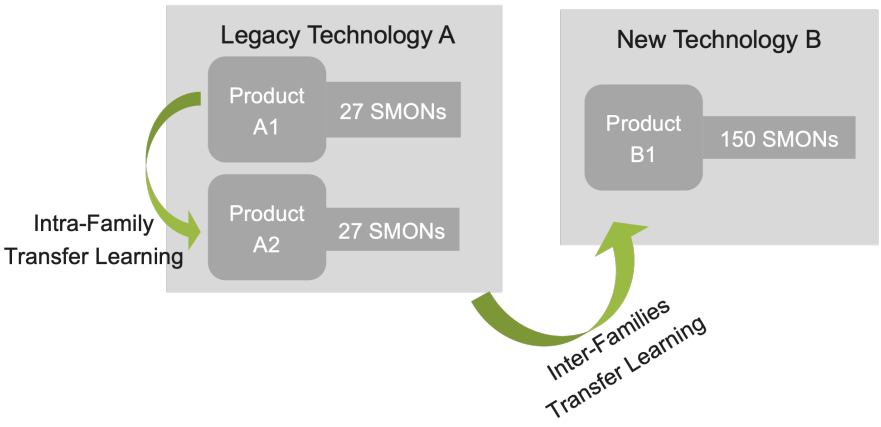


Figure 6.3: The proposed Inter and Intra-Family transfer learning protocol.

The creation of deep learning feature extractors follows a four-step process:

1. The deep learning models are initially trained to perform an alternative task T_i using the unlabeled data from *A1*.
2. Optionally, the deep models can be fine-tuned using the labeled data to adapt the convolutional kernels to the labeled data distribution. This process uses the model trained on the unlabeled data as a starting point, with the pre-trained layers' weights serving as initialization, allowing the training optimization to be re-run on the labeled data in a warm-start fashion.

3. The last layer of the DL models, which links the non-linear feature interactions computed in the previous layers to the task chosen in step 1, is removed. The models then function as data transformers, projecting the data from its original space into a new space where solving the original performance prediction task is more straightforward.

4. A linear model, such as Ridge Regression, is then trained using the features extracted in the previous step as inputs. This model combines these features to solve the performance prediction task, utilizing an L2-regularized Mean Squared Error loss in the case of Ridge Regression.

An example of the proposed approach is presented in fig. 6.4.

The optional nature of step 2 depends on the size of the labeled dataset. Insufficient samples may lead to overfitting, preventing the network from generalizing to new, unseen samples, potentially resulting in worse performance compared to the feature extractor without fine-tuning. However, if the representation learned by the network on *A1* is sufficiently general, this step may not yield significant improvements in prediction performance. In the absence of fine-tuning, the deep models are utilized "as-is," meaning the layers are fixed (or *frozen*) and no parameters are altered. Additionally, hyperparameter tuning is unnecessary since the models are not re-trained during this process, and parameters are inherited from legacy products.

In step 1 of the proposed approach, the DL models are pre-trained on an alternative task T_i using the unlabeled data. Three popular pre-training strategies have been selected and are detailed in the subsequent subsections.

6.3.1 Pseudo-Labeling

Initially, a supervised ML model is constructed using the available labeled data as a prototype. This model is then applied to the unlabeled data to generate a pseudo-label for each unlabeled sample, resulting in an augmented training set $\hat{X}_u = \{(x_{u_1}, \tilde{y}_{u_1}), (x_{u_2}, \tilde{y}_{u_2}), \ldots, (x_{u_n}, \tilde{y}_{u_n})\}$. Each pseudo-label $\tilde{y}_{u_i}$ is directly correlated with the target task, namely, performance prediction. A supervised DL model is subsequently trained on $\hat{X}_u$, allowing the model to learn how to manipulate SMON values to generate features that are useful for the performance prediction task. This model is later fine-tuned using the actual labeled training set. It is expected that models incorporating features obtained via pseudo-labeling will achieve higher accuracy than those relying solely on an unsupervised DL feature extractor.

6.3.2 Auto-Encoder

This task involves reconstructing the initial input, with the training set defined as $\hat{X_u} = \{(x_{u_1}, x_{u_1}), (x_{u_2}, x_{u_2}), \ldots, (x_{u_n}, x_{u_n})\}$, where the input and output of the deep ML model are identical. The model consists of two components: the encoder, which projects the data into an alternate space $\widetilde{X}$, and the decoder, which learns to recover the original data X from $\widetilde{X}$ (refer to fig. 6.4). This task is entirely unsupervised, as only the SMON values X from the unlabeled set are utilized. Upon completion, the entire decoder is discarded, with only the encoder retained as a feature extractor (as shown in fig. 6.4). The resulting feature manipulations $\widetilde{X}$ are subsequently linked to F_{max} through a linear regression model. Given that this task is agnostic to the specific performance prediction objective, the extracted features serve as a more generalized non-linear transformation of the SMON input features.

6.3.3 Denoising Auto-Encoder

This approach is similar to the previous task, with the primary difference being that the inputs to the model are noisy versions of the data, corrupted with superimposed Gaussian noise. The training set is thus defined as $\hat{X_u} = \{(\tilde{x}_{u_1}, x_{u_1}), (\tilde{x}_{u_2}, x_{u_2}), \ldots, (\tilde{x}_{u_n}, x_{u_n})\}$, where $\tilde{x}_{u_i}$ represents a corrupted version of x_{u_i}. The model is trained to recover the original data from these noisy inputs. Similar to the Auto-Encoder task, the model can be divided into an encoder and a decoder, and this process is entirely unsupervised and unaware of the specific problem at hand. After the training, the decoder is discarded, and only the encoder is used as a feature extractor.

6.4 Experimental Evaluation

6.4.1 Setup

The experiments conducted utilized real-world MCU products with the objective of developing models that transcend individual product variations. The methodology proposed herein has been validated using a dataset comprising 2,986 labeled *A1* (section 5.5) devices and 1,015 labeled *A2* devices from product family *A*. The deep neural networks were pre-trained on 1,496,248 unlabeled *A1* samples collected from production lots. Both *A1* and *A2* devices, being part of the same product family, share the same set of 27 SMONs. However, *A2* represents a higher-performance variant with additional cores and increased memory capacity compared to *A1*. The *A1* dataset includes 10 labels for F_{max} across various functional test patterns, while the *A2* dataset provides 20 such labels. For both

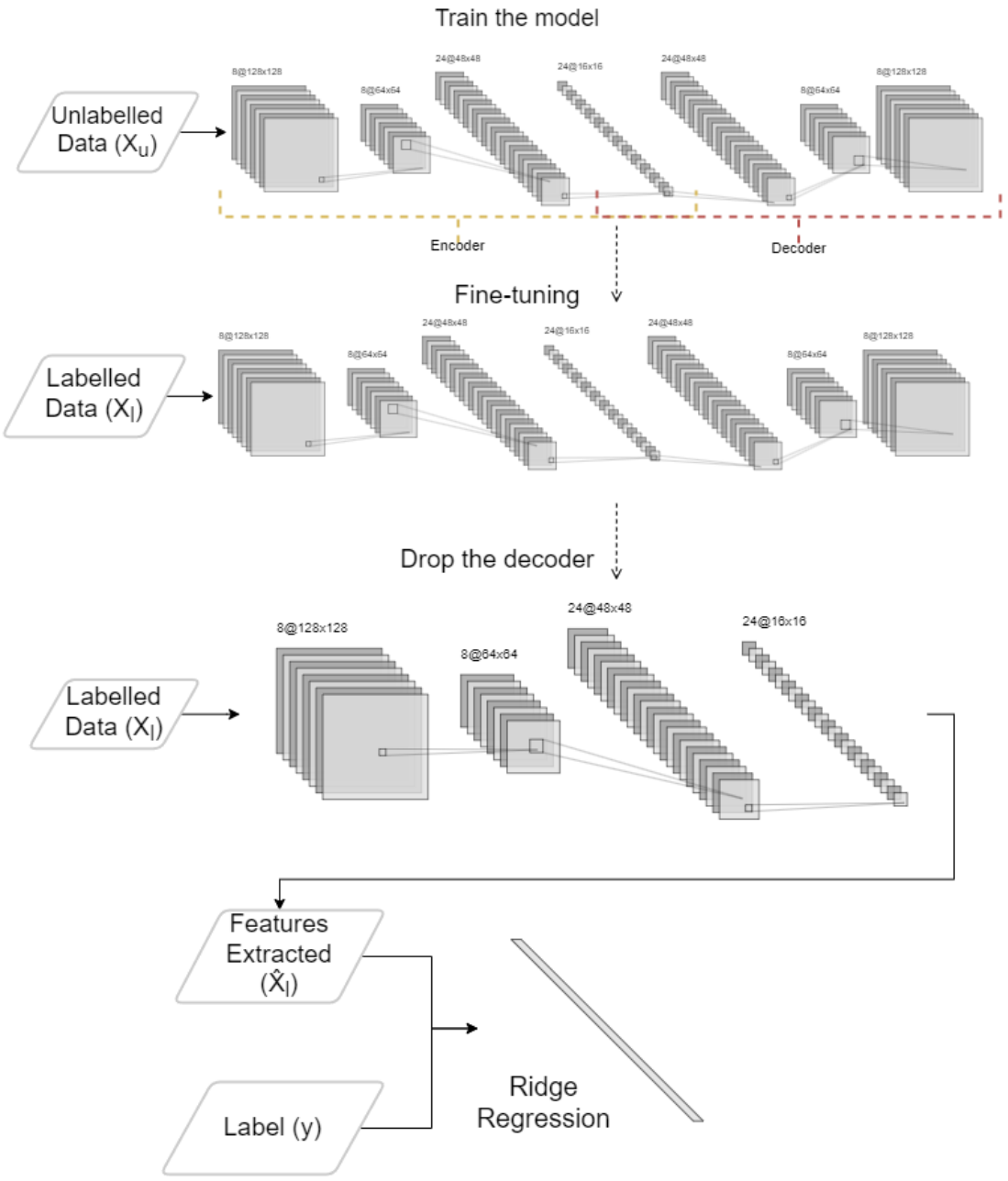


Figure 6.4: An example of training a deep convolutional feature extractor in a Semi-Supervised fashion using the Auto-Encoder alternate task. Initially, the deep model is trained with unlabeled data, and then it is used as a transformer for labeled data (for the same or a different product). Finally, a Linear Model is trained to link the feature embeddings with the labels.

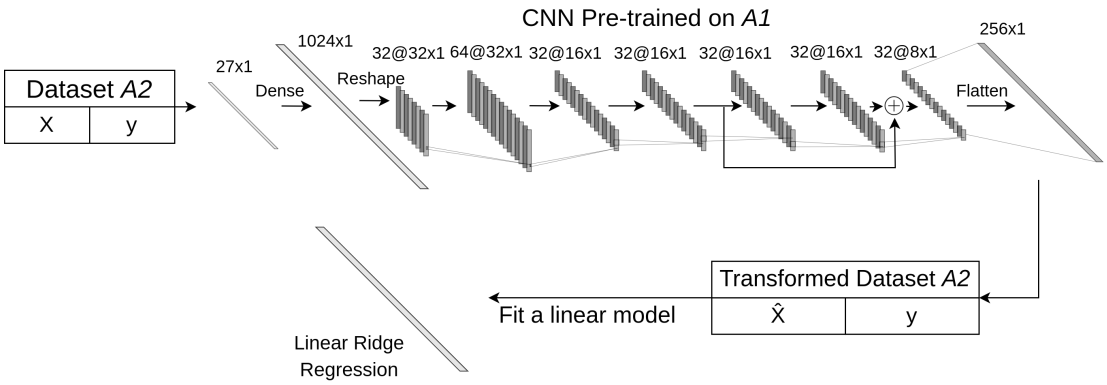


Figure 6.5: An illustration of the proposed transfer learning approach between the *A1* and *A2* products using the PL-CNN. The *A2* dataset is first transformed using the pre-trained network, which remains frozen, with no updates to the convolutional layers. A linear regression model is then trained to link the feature embeddings from *A2* with the $F_{\max}$ labels.

products, the target performance metric used in the model is the artificial label $P_{\min}$, which represents the minimum $F_{\max}$ value among the available labels, as outlined in section 2.5.

For product family B, the dataset consists of 2,002 labeled samples from product $B1$, with 150 SMONs considered during the experiments.

Data preprocessing involved standardizing features by removing the mean and scaling to unit variance.

An initial outlier detection technique was employed [31], utilizing the Interquartile Range (IQR) method [86] applied to the SMON input space. Samples outside the defined range $[lb, ub]$ were deemed outliers and removed from the dataset section 3.8.

A baseline model for comparison was established using Ridge Regression with Principal Component Analysis (PCA) employing 14 components and polynomial transformation of SMON values chapter 5 (referred to as Poly Ridge, PLR, or PCA Poly Ridge). On the previous chapter, it was established that for *A1* polynomial models provide a better fit for the SMONs-$F_{\max}$ relationship compared to linear models. Additionally, comparisons were made with linear models (Ridge Regression, denoted as L. Ridge, and Linear Regression, denoted as L. Reg), as well as Random Forest [50, 44] and KNN-5 [50].

As a baseline evaluation, the Poly Ridge model trained on *A1* was initially applied to *A2* without fine-tuning, highlighting the challenge of reusing models across different products and underscoring the need for model updates. Subsequently, the Poly Ridge model was retrained exclusively on *A2* samples, without

leveraging prior knowledge from *A1*.

The deep feature extraction models derived from *A1* were then applied to new datasets from *A2* and *B1*, resulting in transformed datasets $D^{A2} = (\hat{X}^{A2}, y^{A2})$ and $D^{B1} = (\hat{X}^{B1}, y^{B1})$, where $\hat{X}^{A2}$ and $\hat{X}^{B1}$ represent the projections of X^{A2} and X^{B1} into the latent space learned by the neural network. A Ridge Regression model was subsequently trained on D^{A2} and D^{B1} to map the transformed SMONs to the actual $F_{\max}$ values.

During the testing phase, the SMON values for *A2* and *B1* were first transformed using the deep feature extractor, followed by the application of the Ridge Regression model to predict $F_{\max}$. The sequence of applying DL models $A1 - A2 - B1$ was based on data availability and product similarity. Alternative sequences, involving increasing feature dimensionality to adapt A-SMONs to B-models, may be explored if data permits.

A 5-fold train-test split with a 75%-25% proportion, generated using different random states, was used to construct learning curves, potentially fine-tune the deep feature extractor, and evaluate model performance. Each performance metric is based on the average of results from these five splits, reducing bias in error estimation.

Performance metrics were reported in terms of normalized Root Mean Square Error (nRMSE), normalized Mean Absolute Error (nMAE), learning curves, the Area Under the nRMSE Learning Curve (AUC-nRMSE), and Guardband G computed trough bootstrapping (section 3.7). The learning curves plot the model's prediction performance (nRMSE) against the number of training samples, demonstrating how performance evolves with varying sample sizes. The AUC-nRMSE measures the model's ability to generalize with fewer labeled samples, where a lower AUC-nRMSE indicates better generalization. For the initial 20 points of the learning curves, AUC-nRMSE values were computed, covering a range from 300 labeled samples for $B1$ to 600 for $A1$ and 170 for $A2$.

An additional evaluation metric is the number of samples required to achieve a 2% nRMSE on the test set, indicating a satisfactory error margin and performance screening yield [82].

All experiments were conducted using Python with PyTorch for deep learning models on a server equipped with an Intel® Core™ i9-9900K CPU @3.60GHz x 16, 32GB of RAM, and an Nvidia® 2080 TI GPU.

The code and models developed for this chapters are available in a public GitHub repository[1].

[1] https://github.com/BellaNico4/DL-Strategies-in-MCU-Screening

6.4.2 Inter-Family Transfer Learning Setup

The inter-family transfer learning protocol requires an initial pre-processing step to project the input space of product $B1$ (comprising 150 SMONs) to align with the input dimensionality of the neural network (n, where $n = 27$ in this case). Several techniques were investigated for this projection:

- **PCA Projection:** This method employs Principal Component Analysis (PCA) to transform the SMONs space of product $B1$ to match the dimensionality of the neural network.
- **Random SMONs Sampling:** This technique involves selecting n SMONs randomly (referred to as Dummy-Feature Selection CNN or D-FS CNN). Given the high linear correlation among SMONs, random sampling is a reasonable approach. This method is unbiased and requires no additional information about the types and technologies of SMONs.
- **Repeated SMONs Sampling:** This method extends the previous approach by iteratively sampling n SMONs from the pool of 150 and using them to train an ensemble of networks. An ensemble of 10 Dummy CNN models was created, with random sampling enabling exploration of different regions within the SMONs input space. This technique facilitates distinct feature interactions across models and has been shown to reduce variance in predictions and improve classification accuracy [96]. The final prediction is derived from the average of the 10 outcomes (hard-voting mechanism), resulting in the Dummy-Feature Selection Voting CNN (D-FS V-CNN).
- **Best SMONs Selection:** This feature selection strategy involves choosing the top n SMONs based on their relevance to the performance prediction task. If test engineers have prior knowledge of which SMONs are most indicative of device performance, these can be manually selected for model construction. An RFE-based approach was used to rank SMONs by importance, selecting the top 27 as the BEST-FS strategy. This approach is considered a 'gold standard' for comparison, as the importance is determined from the entire labeled dataset [28].
- **Correlation Approach:** Following the method outlined in [50], this approach selects the n SMONs with the highest Pearson correlation coefficients with the target $F_{\max}$. This method is denoted as Corr-FS.

The reduction in model complexity (in terms of parameters or features analyzed) often correlates with a decrease in the number of labeled samples required

to achieve a specified level of generalization error. Accordingly, comparative experiments involving the aforementioned feature reduction techniques (PCA, Best-FS) applied to Polynomial Ridge models will also be presented.

Details regarding the architectures and training setup for the CNNs utilized are provided in the subsequent section section 6.4.3.

6.4.3 Deep Learning Models

The following deep learning (DL) models were trained and evaluated on dataset $A1$, as outlined in section 7.4. Only the most effective models were subsequently applied to datasets $A2$ and $B1$ for Transfer Learning.

1. **Soft-Ordering 1D Convolutional Neural Network (PL-CNN):** This model consists of a one-dimensional convolutional neural network (1D-CNN) that begins with a fully connected layer, utilizing the CELU activation function [14]. This initial layer projects the input features into a higher-dimensional space through a non-linear transformation of the SMONs, as depicted in fig. 6.5. This transformation allows the subsequent 1D convolutional layer to detect relevant interactions among the tabular features, even though these features are not spatially correlated as they are in images [13]. The output is reshaped into image-like samples with dimensions $(H, 1, C)$, where H represents height, 1 denotes length, and C indicates the number of channels. Each sample is a group of H ordered features with C different permutations. The output is then flattened before being input into the final supervised learning model.

2. **Fully-connected Autoencoder (AE-FC):** This model is composed of fully connected layers that perform feature space augmentation without using a bottleneck, thereby projecting the data into a higher-dimensional space.

3. **Denoising Fully-connected Autoencoder (DAE-FC):** Similar to AE-FC, but this model is trained using a corrupted version of the original input data to enhance its robustness.

4. **Convolutional Autoencoder (AE-CNN):** Incorporating soft-ordering, this model utilizes a 1D convolutional encoder and a de-convolutional decoder. The encoder facilitates feature space augmentation, and the output is flattened prior to being input into the final supervised model, as shown in fig. 6.5.

5. **Denoising Convolutional Autoencoder (DAE-CNN):** Similar in structure to the AE-CNN, but trained with data that has been corrupted by Gaussian noise. The network is tasked with reconstructing the original SMONs

values from the noisy input.

Each convolutional layer is preceded by a Batch Normalization layer and followed by a Dropout layer to improve generalization. The training configuration for these DL models on dataset *A1* consisted of using 85% of the dataset for model training, with the remaining 15% reserved for validation purposes. Training continued for a maximum of 100 epochs or until there was no improvement in validation performance over the last 10 epochs, triggering early stopping [36]. The optimization was conducted using the Stochastic Gradient Descent (SGD) method [36], with a learning rate (lr) initialized at 1×10^{-4} and momentum set to 0.7 [36]. The learning rate was reduced incrementally upon reaching a plateau in loss, down to a minimum of 1×10^{-5}. This strategy was also employed for fine-tuning the deep models, as indicated in the tables within section 6.4.4. In cases where full DNN fine-tuning was not executed, only the Batch Normalization layers were updated to adapt to new data distributions presented by datasets $A2$ and $B1$.

6.4.4 Experimental Results

This section provides an analysis of the results obtained from deep learning-based approaches for MCU performance prediction. The discussion begins with the application of semi-supervised techniques to develop deep feature extractor models using dataset $A1$. These models are evaluated in comparison to a baseline shallow learning model (Polynomial Ridge), with a focus on prediction performance, measured by error percentage, and the reduction in the number of labeled samples required to achieve comparable performance. Following this, the most effective deep feature extractor models are identified for implementing a transfer learning framework between devices from datasets $A1$ and $A2$. Finally, the outcomes of applying inter-family transfer learning to the samples from dataset $B1$ are presented.

Semi-Supervised Learning on $A1$ Product

The experiments reveal that nearly all DL models effectively harness relevant information from unlabeled data, achieving prediction errors comparable to those of the baseline Polynomial Ridge model (table 6.1 and fig. 6.6). Linear models, such as SVMs, Decision Trees, and k-Nearest Neighbors (KNN), exhibited substantial limitations, evidenced by higher prediction errors and reduced AUC values; hence, they are excluded from fig. 6.6. Non-linear models, including KNN-5 and Random Forest (RF), produced reasonable prediction errors but failed to match the performance of the baseline model.

The fully connected autoencoder (AE-FC) also demonstrated its suitability by providing effective feature extraction for the given objectives.

Among the DL models, the CNN-PL model emerged as the most effective. It achieved a notably lower normalized root mean square error (nRMSE) and guardband compared to the baseline model (1.48% vs. 1.57% nRMSE, 9.43% vs. 10.04% guardband, as shown in table 6.1). The reduction in guardband is particularly significant for enhancing production yield. Furthermore, with a minimal amount of labeled data (89 samples for PL-CNN), the model maintained an nRMSE consistently below 2%, with the learning curve leveling off, indicating limited error reduction beyond this point (see fig. 6.6). This neural network effectively identifies pertinent features for the regression task and generalizes well to new, unseen data, even with a limited training dataset. Achieving performance comparable to the baseline Polynomial Ridge model (1.57% nRMSE) with just 500 labeled samples—compared to 2986 samples required for the baseline—illustrates a significant reduction in effort for MCU characterization and dataset creation. Labeling each sample requires at least 30 minutes; thus, achieving 1.57% nRMSE with the Polynomial Ridge model would necessitate approximately 89,580 minutes (or 63 days) of continuous labeling. In contrast, the CNN-PL approach reduces this time to less than one-sixth (approximately 10 days).

This efficiency is also noted across other DL models, where fewer samples are required to achieve target prediction errors, although the reduction in training set size is less pronounced (table 6.1). Nevertheless, all DL models achieved errors below 2% (indicated by the dotted line in fig. 6.6) with a limited number of samples (89 for PL-CNN, AE-FC, and DAE-FC; 119 for AE-CNN; and 149 for DAE-CNN). The superior performance of the PL-CNN model is attributed to its tailored feature extraction for the specific prediction task, unlike other methods that apply more general feature transformations. Consequently, the PL-CNN model will be utilized as the primary feature extractor in subsequent experiments involving datasets *A2* and *B1*. For comparison, the AE-CNN model will be applied to dataset *A2*.

Transfer Learning on the *A*2 Dataset

Results illustrated in table 6.2 demonstrate that applying a model trained on *A1* directly to *A2* yields a notably high prediction error (first row, 33.93% nRMSE). It is therefore necessary to adjust the model coefficients to accommodate the new product. Retraining the Polynomial Ridge model specifically on *A2* proves effective, achieving satisfactory accuracy (mean nRMSE of 1.54% with a standard deviation of 0.11%, across five folds) when the complete training dataset is utilized. However, this retraining process requires a sufficiently large training dataset.

The performance achieved by employing deep feature extractors surpasses that of the baseline in terms of both nRMSE and guardband (mean nRMSE of 1.51% with a standard deviation of 0.13% vs. 1.54% mean nRMSE with a standard deviation of 0.11%, 11.01% mean guardband with a standard deviation of 1.05%

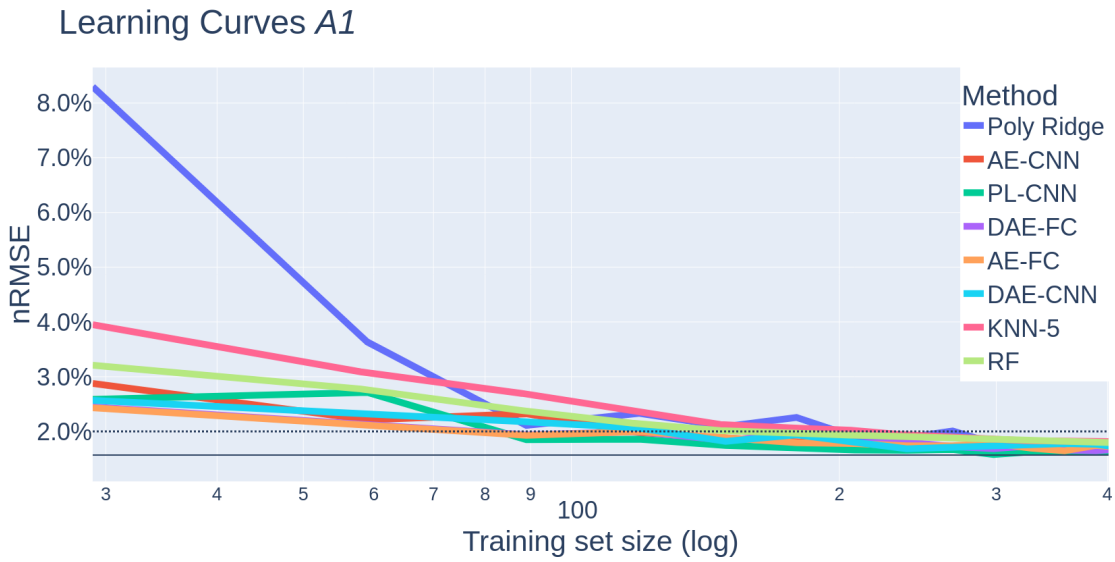


Figure 6.6: Learning curves for different deep and shallow models. The x-axis represents the number of training samples (log scale, truncated at 400 samples), and the y-axis represents the nRMSE on the test set. The upper dotted line denotes 2% nRMSE, while the lower line indicates the baseline Polynomial Ridge prediction error obtained with the entire training set (1.57% nRMSE).

Table 6.1: Prediction performance metrics of deep feature extractors on product *A1*

Model	nRMSE	nMAE	G	Samples to 1.57% nRMSE	AUC
Polynomial Ridge [20]	1.57%	1.17%	10.04%	2,986	12.1
Linear Ridge [44, 50]	2.60%	1.95%	16.54%	–	15.1
Linear Regression [50]	2.60%	1.95%	16.55%	–	19.1
RF [44, 50]	1.63%	1.22%	10.31%	–	11.0
KNN-5 [50]	1.69%	1.27%	10.31%	–	11.5
PL-CNN	**1.48%**	**1.09%**	**9.43%**	**477**	**9.83**
AE-FC	1.56%	1.15%	9.96%	1,702	10.1
AE-CNN	1.52%	1.13%	9.70%	1,971	10.4
DAE-FC	1.55%	1.14%	9.95%	1,702	10.1
DAE-CNN	1.54%	1.14%	9.85%	1,822	10.4

vs. 11.19% mean guardband with a standard deviation of 0.85%, using the PL-CNN model, see table 6.2). These superior outcomes are obtained with a smaller

portion of labeled data: 227 samples are necessary to achieve a 1.54% nRMSE using PL-CNN, and only 16 samples are required to reach a 2% nRMSE (fig. 6.7).

Experimental results indicate that feature extractor models initially trained on *A1* are capable of identifying relevant features in *A2*, even without fine-tuning (FT, as denoted in the plots) of the intermediate convolutional layers. This suggests that the feature manipulation performed by neural networks is sufficiently general to be effectively applied across different products. Fine-tuning of intermediate layers does not yield further improvements, likely due to the limited amount of training data available from *A2*, which is insufficient for adequately updating the entire network (with parameters numbering in the tens of thousands), leading to overfitting.

The reduction in the required amount of labeled data decreases the effort involved in the MCU characterization phase and data collection, thus reducing the time necessary to acquire an adequate dataset for ML models. Labeling each sample requires at least 30 minutes. By utilizing the PL-CNN model, the time required to build and deploy an ML-based predictive model for the operating frequency of a new MCU is reduced to just $30 \text{ min} \cdot 16 \text{ samples} = 480 \text{ min} \approx 8$ hours. Additionally, no further feature engineering is necessary due to the neural networks' capability to autonomously extract relevant features.

Table 6.2: Prediction Performance Metrics of Intra-Family Transfer Learning on Product *A2*

Model	nRMSE	nMAE	G	Samples to 2% nRMSE	AUC
Poly Ridge (*A1*)	33.93%	33.63%	77.38%	–	-
Poly Ridge (*A2*)	1.54%	1.16%	11.19%	121	3.26
PL-CNN	**1.51%**	**1.13%**	**11.01%**	**16**	**2.54**
PL-CNN (FT)	1.52%	1.14%	11.05%	24	2.6
AE-CNN	1.54%	1.15%	11.21%	24	2.65
AE-CNN (FT)	1.74%	1.33%	12.68%	40	2.94

Transfer Learning on the $B1$ Dataset

The inter-family transfer learning protocol is implemented by transferring knowledge from dataset A to dataset B. The PL-CNN model, noted for its exceptional performance in previous tasks, is utilized for this transfer. Comparisons are made with alternative models, including Polynomial Ridge, k-Nearest Neighbors (KNN-5) [50], and Random Forest [50, 44].



Figure 6.7: Learning curves for different models on the *A2* product. The x-axis represents the number of training samples (log scale), while the y-axis represents the nRMSE computed on the test set. The upper dotted horizontal line corresponds to 2% nRMSE, and the lower horizontal line indicates 1.54% nRMSE, which is the final error obtained by the Polynomial Ridge. The use of Transfer Learning and Deep Feature Extractors allows for lower prediction error with a smaller training set.

As illustrated in fig. 6.8, the learning curves for the $B1$ product demonstrate that the CNN model, pre-trained on a large dataset, achieves a lower prediction error compared to the baseline Polynomial Ridge model and requires fewer labeled samples for effective training.

Feature reduction techniques applied to the Polynomial Ridge model, specifically PCA and BEST-FS, have shown to be advantageous. The BEST-FS Polynomial Ridge model exhibits a more favorable learning curve compared to the Polynomial Ridge models with and without PCA. Reducing model complexity by decreasing the number of parameters generally benefits model performance by lowering the amount of labeled data needed to achieve a specified generalization error. Simplified models are less prone to overfitting, particularly when the training dataset is limited. However, balancing model complexity is crucial: this issue is commonly referred to as the "bias–variance tradeoff" [65]. Reducing parameter variance across samples often involves increasing bias, which can lead to underfitting if the model fails to capture important relationships between features and target outputs. Conversely, high variance can occur if the model overfits the training data by modeling its noise.

Among the evaluated models, CNNs exhibited the lowest error rates in the initial stages of the learning curve, demonstrating robust generalization capabilities with a limited training set. The highest performance was attained using an ensemble of Dummy-CNN models, which effectively mitigates model variance and enhances prediction accuracy [65]. Although Random Forest models achieved comparable performance to CNNs during the early learning phases, CNNs ultimately provided lower prediction errors when trained on the entire dataset.

CNNs demonstrated sufficient generalization capability to enable the transfer of knowledge from product A to product B, establishing a feature representation that is applicable across different product families. Integrating information from the most effective SMONs into the Transfer Learning Protocol facilitated excellent generalization with a minimal number of samples, as evidenced by the performance of the Best-FS CNN model. In cases where the optimal SMONs for $B1$ are not identified, it becomes essential to reduce the dimensionality of the input data for the pre-trained CNN. In this regard, PCA proved to be more effective than Dummy-FS, as it efficiently compresses the information from all available SMONs. To thoroughly explore the feature space, particularly with the larger number of SMONs, the ensemble approach using the Dummy-FS Voting CNN outperforms both the PCA method and the Polynomial Ridge model with BEST-FS. As highlighted in table 6.3, the AUC-nRMSE metrics indicate that the Dummy-FS Voting CNN represents a strong alternative, achieving an AUC value marginally higher than that of the Best-FS CNN, which generally exhibits the best performance.

6.5 Limitations

Deep and semi-supervised learning techniques can reduce labeling effort by leveraging feature transformations learned from production data. However, the dimensionality of these models must be carefully considered. Neural networks are inherently more complex than simpler models, such as linear regression with or without polynomial transformations. While lightweight regression models can be deployed directly onboard for real-time performance screening, neural networks may not be suitable for such applications due to their computational demands.

Moreover, designing and implementing neural networks involves a more intricate engineering process, primarily because of the higher number of hyperparameters that need tuning. This complexity necessitates additional expertise and resources during development.

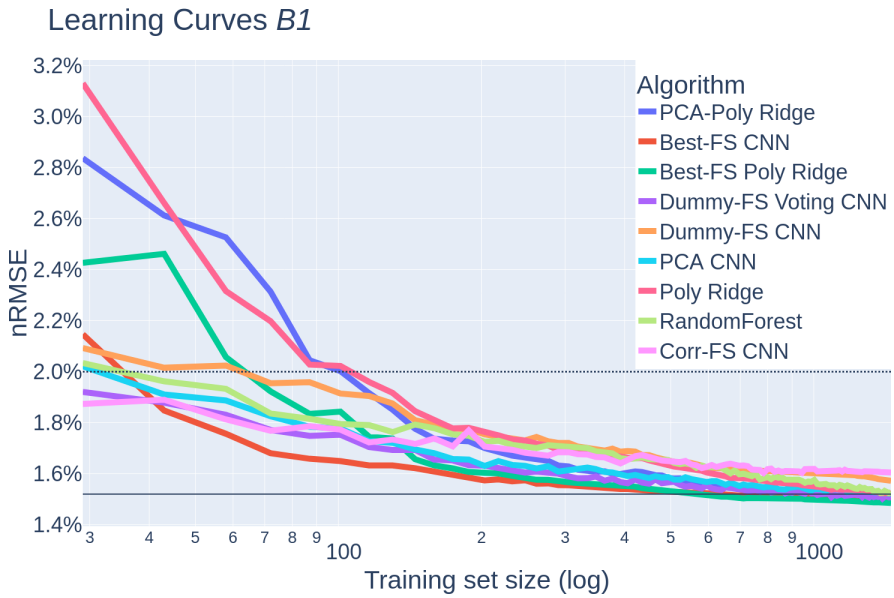


Figure 6.8: Learning curves for different models on product $B1$. The x-axis represents the number of training samples (log scale) while the y-axis shows the nRMSE computed on the test set. The upper dotted horizontal line corresponds to 2% nRMSE, and the lower horizontal line represents 1.52% nRMSE, the final error achieved by the Polynomial Ridge model. Model simplification (via FS) and the application of Transfer Learning reduce the number of labeled samples required for model training.

6.6 Conclusion

A deep learning-based framework has been introduced in this chapter to optimize MCU performance screening using DL techniques. This methodology, grounded in Semi-Supervised and Transfer Learning, effectively minimizes prediction errors while also significantly reducing the number of labeled samples required—by as much as sixfold—for constructing predictive models in the product family A. By leveraging the available unlabeled samples, deep neural networks have been developed to serve as feature extractors, projecting data into a higher-dimensional space. In this space, a simple linear Ridge Regressor can accurately predict device performance. The improved accuracy in prediction facilitates a reduction in guardband G, thereby enhancing process yield by decreasing the number of functional devices erroneously discarded.

The deep learning models, pre-trained on the legacy product $A1$, offer a foundational base for developing models for MCU products within the same or different

Table 6.3: Average Results on $B1$ for 5 Different Training-Test Splits.
*Best Feature Selection (B-FS) is considered a "gold standard" for comparison, as the importance of each SMON was determined using the entire labeled set.

Model	nRMSE	nMAE	G	Samples to 2% nRMSE	AUC nRMSE
B-FS PLR*	1.49%	1.15%	9.04%	63	4.82
PCA-PLR [44, 20]	1.50%	1.16%	9.16%	106	5.24
PLR	1.52%	1.17%	9.26%	101	5.35
KNN-5 [50]	1.65%	1.27%	10.07%	87	5.36
RF [50, 44]	1.53%	1.17%	9.30%	35	4.91
B-FS CNN*	1.49%	1.16%	9.08%	35	4.52
D-FS V-CNN	**1.50%**	**1.16%**	**9.17%**	**29**	**4.66**
PCA CNN	1.52%	1.17%	9.25%	31	4.73
D-FS CNN	1.57%	1.21%	9.55%	63	5.06
Corr-FS CNN	1.60%	1.24%	9.76%	29	4.8

families, irrespective of variations in SMONs. This capability supports the application of Transfer Learning to new products ($A2$) or even across product families (B). The employment of Deep Feature extractors during the knowledge transfer process contributes to a reduction in the size of the required training set. Consequently, the decrease in labeled data requirements alleviates the workload during the MCU characterization phase and data collection, thus expediting the creation of an appropriate dataset for ML models.

With the application of carefully selected techniques, only a minimal number of samples are necessary to construct and deploy an ML-based predictive model for MCU performance screening. For instance, in product ($A2$), the use of deep feature extractors reduces the labeled sample requirement to one-fourth of what is needed to achieve equivalent error rates with a baseline Poly Ridge model. Additionally, a mere handful of samples (approximately 15) suffice to attain acceptable levels of prediction error and guardband (2% nRMSE, 9%-10% guardband). Similarly, for product ($B1$), satisfactory prediction error and guardband can be achieved with fewer than 30 samples.

The SSL/Transfer Learning framework developed here is versatile and can be applied to various scenarios characterized by a large volume of unlabeled data, such as MCU performance screening or alternate testing. Moreover, the feature extraction process— and, by extension, the transfer learning strategy—outlined

in this study is applicable to other domains where pre-trained models are available, allowing for the reuse of existing knowledge in different but related fields, particularly when labeled samples are scarce.

Chapter 7

Feature Selection

In the preceding chapters, both shallow and deep learning models were developed to support performance screening of MCUs using ML. Specifically, chapter 6 addressed the complexities associated with adapting ML models to newly released MCU products, and this challenge was addresed by using an SS/Transfer Learning protocol with deep learning. In this context, the introduction of product family B is noteworthy, as it features a greater number of SMONs compared to family product A (150 SMONs in $B1$ versus 27 in $A1$ and $A2$). Additionally, the number of SMONs can vary across different products within family B.

The rationale for an increased SMON count lies in the correlation between the number of SMONs and device speed information. A larger set of SMONs is expected to provide more comprehensive data, potentially improving the accuracy of ML models.

Moreover, having numerous SMONs might enable the approximation of the SMONs-$F_{\max}$ relationship using linear models, thereby reducing the relevance of previous feature transformation steps (such as those involving deep learning or polynomial features) concerning final ML model prediction errors. However, challenges emerge when the SMON count exceeds a certain threshold. This issue, often referred to as the curse of dimensionality (CoD), arises because high-dimensional spaces cause data points to become increasingly sparse, complicating model accuracy and increasing the number of samples required for effective ML model training [76]. High dimensionality also escalates computational complexity, exacerbates overfitting, and degrades predictive performance. Additionally, handling extensive feature sets demands substantial data storage and processing resources, affecting both the training and inference phases of ML models. The increased parameter count associated with a high number of features can lead to over-parametrization, risking overfitting [61, 40]. In linear system identification, this is manifested as redundant parameters that deviate from the actual system's physical behavior. From a technological perspective, increasing the SMON count requires additional

silicon space, which in turn can lead to increased current leakage and higher manufacturing costs [28].

This chapter is dedicated to feature selection techniques aimed at reducing the number of features necessary for ML model development, thus minimizing the number of sensors required on the MCU. Various methods will be discussed, including ranking features based on their importance in supervised performance prediction tasks through iterative feature selection, as well as utilizing Compressed Sensing (CS) and regularized linear models for optimizing ML models in MCU performance screening. These approaches significantly decrease feature space dimensions while maintaining prediction performance, thus conserving silicon area. Additionally, the methodology provides valuable insights for test engineers on the correlation between sensors and $F_{\max}$, aiding in diagnostic processes.

CS and regularized linear models are proposed as a cohesive framework for both constructing ML models and performing feature selection. By eliminating irrelevant or non-informative features/SMONs in the context of MCU performance screening, the aim is to reduce the model's footprint, specifically in terms of the number of coefficients/features, while preserving satisfactory prediction accuracy.

Feature selection techniques are generally classified into three categories: filtering, wrapper, and embedded approaches [67]. This chapter emphasizes the application of Compressed Sensing (CS) principles within the embedded feature selection framework, aiming to enhance sparsity in model coefficients. The focus is on linear methods that incorporate regularization techniques, which introduce penalty terms into the error metric used for model development.

The primary objectives of this chapter are:

- Justify the rationale for increasing the number of SMONs on MCUs.
- Illustrate how Compressed Sensing theory, when applied through regularized linear models, can be effectively utilized for performance prediction screening.
- Demonstrate that regularized linear models can reduce the overall model size while significantly preserving prediction accuracy.
- Extend this framework to various MCU products, even when the relationship between SMONs and $F_{\max}$ is not strictly linear.
- Evaluate the impact of both convex and non-convex penalty terms in regularized linear models and assess their comparative effectiveness.
- Compare the performance of regularization techniques against filtering and wrapper methods for feature selection.

7.1 Motivation

As discussed in preceding chapters, SMONs, which are on-chip ring oscillators, serve as critical features for predicting MCU performance. Consequently, both the structure and sensitivity of SMONs have a substantial impact on the accuracy of performance models. To optimize these models, it is advantageous to integrate a diverse array of SMON types into the MCU design. Furthermore, spatial distribution of SMONs is essential to account for within-die (WID) process variations, which become more pronounced with advancing technology nodes [10]. Therefore, there is a positive correlation between the number of SMONs on the MCU and the volume of speed-related data that can be extracted.

Ideally, maximizing the number of SMONs would enhance performance predictions. However, this approach is not without its challenges:

1. From a technological perspective, incorporating a large number of SMONs is problematic as they are used solely for testing purposes and offer no functional benefit to end users. Each additional SMON consumes physical die space, increases current leakage, and elevates production costs. Consequently, a substantial number of SMONs results in significant area overhead and heightened leakage currents.

2. From a machine learning (ML) standpoint, increasing the number of SMONs leads to a higher dimensional feature space for ML models. This escalation can introduce the *curse of dimensionality* (CoD) [76], requiring a larger training dataset to achieve effective model training. Additionally, high-dimensional feature spaces can exacerbate overfitting, particularly when the volume of labeled data is limited, reducing the model's accuracy on new samples.

Moreover, scaling ML-based performance screening presents significant storage challenges. Storing thousands of features for data logging across millions of devices could result in substantial storage requirements.

Despite these challenges, experimental results demonstrate that a higher number of SMONs enables better approximation of the SMONs-$F_{\max}$ relationship using linear models. These models are favored for their simplicity, interpretability, and efficiency, and may even be suitable for deployment on-board for in-field evaluations. Thus, a critical trade-off must be managed: while increasing the number of SMONs provides more information about device speed, it also incurs additional costs. Therefore, minimizing the number of SMONs while retaining predictive accuracy is essential during the MCU design phase.

The rationale for reducing the number of features is evident: technologically, minimizing the number of features lowers production costs by incorporating only the most informative sensors into future devices.

To tackle the challenges associated with high-dimensional datasets, two main strategies can be pursued: enhancing the quantity and quality of training samples or reducing the dimensionality of the dataset. Acquiring a large labeled dataset is often prohibitively expensive and time-consuming, typically limiting the number of samples to hundreds or thousands at most (section 2.5). While techniques such as data augmentation through imputation and filling missing values (section 5.1), Transfer Learning (chapter 6), and Active Learning (section 5.5.5) can potentially enhance the diversity and quantity of training samples, their effectiveness is contingent upon having a sufficient volume of labeled data. Unfortunately, the dataset sizes often fall short of what is required for training complex models, such as generative models, which demand extensive labeled datasets for effective learning.

While data from production lines (chapter 6) can be leveraged, such data may not be available during the preliminary stages of model development for new products. Dimensionality reduction methods like PCA are useful but do not directly reduce the number of SMONs required on the MCU board. Consequently, FS becomes crucial.

Given the preference for linear models due to their simplicity and interpretability, and the necessity for FS to manage dataset dimensionality, Compressed Sensing (CS) emerges as a robust solution. CS provides an elegant method for feature selection within linear models by leveraging the sparsity assumption inherent in real-world datasets. It aims to retain only the most informative features while discarding redundant or less relevant ones.

By integrating CS with linear models, the feature selection process is embedded within the model training procedure, simplifying the workflow and ensuring that the resulting model remains both interpretable and lightweight. This approach aligns well with the use of linear models in this context. Modeling the relationship between SMONs and $F_{\max}$ linearly, while utilizing CS, allows for addressing the aforementioned challenges efficiently. This framework can be applied independently to the MCU product under analysis.

Each product necessitates individual analysis due to variations in SMON sets—both in number and type. Some SMON sets may exhibit high linear correlation among features (fig. 7.1), making certain features less critical for ML models. Features that are highly linearly correlated can provide redundant information, which can be indirectly inferred from other features.

Moreover, the relationship between SMONs and $F_{\max}$ may not always be strictly

linear. In such cases, prior preprocessing of SMONs, such as polynomial transformation, may be required. Linear models can still be effectively applied within this transformed feature space, where the relationship between transformed SMONs values and $F_{\max}$ is linear.

Polynomial transformations increase dataset dimensionality, with the number of polynomial features growing polynomially with the number of input features and exponentially with the polynomial degree. This can result in a significant increase in the number of model coefficients, complicating interpretation and computation. However, CS and regularized linear models address these issues by promoting sparsity in the coefficient space. This approach reduces the number of coefficients, thereby decreasing model complexity, enhancing interpretability, and improving computational efficiency.

7.2 MCU products

In this chapter, a new dataset, designated as $B2$, is introduced. This dataset comprises approximately 400 labeled samples. Unlike the previously discussed product $B1$ (chapter 6), which also belongs to family product B, the MCU product associated with $B2$ features a larger number of Ring Oscillators (ROs), specifically 579 SMONs. For clarity, the dataset will be referred to as B throughout this chapter.

For comparative purposes, product $A1$, which includes 27 SMONs and around 3000 labeled samples in the training set, is also utilized.

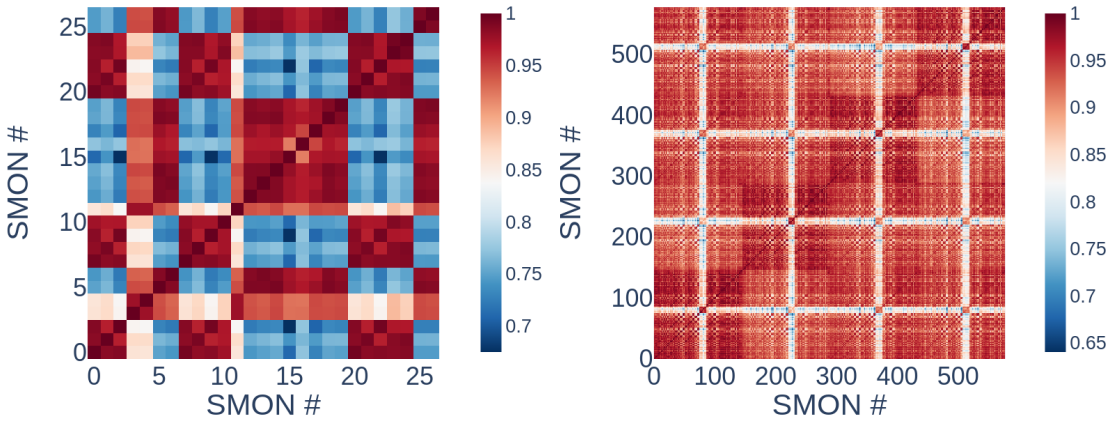


Figure 7.1: Spearman Rank Correlation matrices for two products: A, with 27 SMONs (left) and B, with 579 SMONs (right). In B, SMONs are highly correlated. In A, clusters of features can be identified.

7.3 SMONs module

To address the challenges associated with an increasing number of SMONs on the MCU, a specialized SMON module has been developed. This module integrates a diverse array of SMON types to ensure comprehensive coverage. Identical SMON modules are strategically positioned at multiple locations across the MCU to guarantee adequate spatial distribution.

In addition to these SMON modules, the MCU may also incorporate several functional path ROs, as introduced by Kilian et al. [81]. These functional path ROs (fpath) leverage existing functional paths within the MCU design as ROs, thus eliminating the need for additional dedicated ROs on the silicon.

The combination of features from various SMON modules and functional path ROs provides a robust basis for characterizing chip behavior across all areas and offers substantial benefits for data processing. Consequently, the MCU may ultimately be equipped with thousands of sensors, organized into several identical SMON modules.

Since the SMONs within each module are identical, their varying contributions are solely attributable to their spatial placement on the chip.

7.4 Comparative analysis

As detailed in chapter 3, alternative approaches to embedded FS and CS include filtering and wrapper methods. However, these approaches present notable limitations. Filtering methods are constrained by their reliance on univariate metrics, evaluating individual features in isolation without accounting for interactions between features [67]. Furthermore, such methods are often implemented in an unsupervised manner. In contrast, wrapper methods, while potentially more precise, are highly time-intensive as they necessitate the training of multiple machine learning models across various feature subsets and subsequent selection of the optimal set via cross-validation [68]. This approach is also susceptible to overfitting and may struggle with highly correlated features [79].

Embedded FS methods, in contrast, integrate feature pruning directly within the model training process, thereby avoiding additional computational overhead. A comparative analysis was conducted to evaluate the effectiveness of regularized Linear Models (which incorporate embedded FS and CS) relative to filtering and wrapper techniques. Specifically, the analysis involved a filtering method based on hierarchical clustering, utilizing Ward's linkage [148] to address multicollinearity by examining Spearman rank-order correlations among features [121]. Hierarchical clustering aims to group similar items into clusters based on a hierarchy, starting with each data point as its own cluster and progressively merging them according

to a distance metric until a single cluster is formed. The two primary hierarchical clustering approaches are agglomerative, which merges individual data points into clusters, and divisive, which recursively splits a single cluster into smaller subsets. Ward's method, a specific linkage criterion, merges clusters to minimize the increase in within-cluster variance, making it particularly effective for datasets with varying cluster variances.

In addition to hierarchical clustering, wrapper methods such as Recursive Feature Elimination (RFE) with a Random Forest as the baseline estimator were employed for comparison [57]. This analysis assesses the reduction of multicollinearity within the dataset and evaluates the performance of the various methods in downstream regression tasks by measuring several prediction metrics on a dedicated test set.

When utilizing the Ward distance method, selecting an appropriate distance metric for calculating dissimilarity between data points, such as Euclidean or Manhattan distance, is crucial, as it impacts the clustering outcome.

Initially, the Spearman correlation matrix is transformed into a distance matrix by computing its complement to 1 (i.e., a lower distance signifies a higher correlation among features). Subsequently, hierarchical clustering is performed using Ward's linkage criterion.

A dendrogram is employed to visualize the clustering results. Each leaf in the dendrogram corresponds to an individual data point, while branches represent clusters formed by data points with greater similarity. This visualization facilitates an understanding of cluster relationships and aids in determining the optimal threshold for cutting the dendrogram to achieve the desired number of clusters.

By examining the dendrogram, a threshold t is selected to ensure at least one feature from each cluster is retained. The subset of features identified through this process is then utilized for training machine learning models. For instance, in fig. 7.2, $t = 0.04$ for product A and $t = 0.3$ for product B result in the selection of 5 features (out of 27) for product A and 10 features (out of 579) for product B. The relatively small number of features selected for product B reflects the high degree of multicollinearity within its dataset. This unsupervised analysis, based solely on Spearman correlation, is complemented by subsequent feature selection using supervised learning models, particularly those employing L_1 regularization such as LASSO, to further refine the feature set based on relevance to the predictive task. This combined approach is referred to as *CorrT*.

7.4.1 Wrapper Methods

An RFE procedure combined with internal 5-fold cross-validation was employed to ascertain the optimal number of features. A Random Forest [**Breiman2001**]

served as the baseline estimator for this process, a choice supported by its successful application in various studies [57, 68]. Random Forests are advantageous as baseline estimators due to their ability to handle non-linear relationships between features and targets, obviating the need for prior non-linear transformations. Additionally, Random Forests inherently provide feature importance scores, which facilitate feature ranking and selection in the RFE process. The model's ease of use and established efficacy in previous research [50, 44, 20, 102] further justify its selection as a baseline.

However, despite its merits, Random Forests were not utilized as the final estimator for several reasons. First, their interpretability is limited; being an ensemble method comprising multiple decision trees, Random Forests are less transparent compared to simpler models such as linear regression. Second, the computational demands of deploying numerous decision trees can be significant, making Random Forests less suitable for real-time applications compared to more straightforward linear models. Third, Random Forests often have high memory usage, especially with many or deep trees, which poses a limitation in memory-constrained environments. Lastly, Random Forests are prone to overfitting, particularly when labeled data is limited, as is the case with dataset *B*. Given these considerations, simpler Linear Models are preferred for deployment, especially when similar predictive accuracy can be achieved.

The RFE procedure yielded 23 features for dataset *A* and 359 features for dataset *B*.

7.5 Experimental Evaluation

The forthcoming section outlines the results obtained from the application of regularized linear models and the pre-defined feature selection methodologies. Initially, the experimental framework is detailed, encompassing the datasets, employed models, and evaluation metrics. Following this, an analysis is conducted on various feature sets derived through the three examined feature selection strategies, with a focus on their effectiveness in addressing multicollinearity within the datasets. The section concludes with an empirical assessment of how these approaches perform in relation to the regression tasks for datasets *A* and *B*.

7.5.1 Experimental Setup

As detailed in section 7.2, the proposed methodology was evaluated using two distinct datasets derived from MCUs with varying specifications regarding ROs,

maximum achievable frequency, core count, and memory. The first dataset, referred to as *A1*, comprises approximately 3,000 labeled samples, previously examined in chapters 5 and 6. The second dataset, denoted as B, includes about 400 labeled samples.

In dataset A, each MCU is equipped with 27 SMONs, whereas dataset B features a considerably larger array of 579 SMONs.

Both datasets provide ten labels corresponding to measurements of F_{max} across various functional test patterns, with the ultimate performance indicator for the model being the artificial label P_{min} (section 2.5).

For pre-processing, features are standardized by centering and scaling to unit variance.

The machine learning models employed, as outlined in chapter 3, include Polynomial Ridge as the baseline, along with linear regression models incorporating both convex and non-convex regularization.

For dataset A, prior analyses (chapters 5 and 6) indicated a polynomial relationship between the target and the features. Consequently, a degree-2 polynomial transformation of the SMONs values was applied for training models. For models incorporating prior feature selection (i.e., CorrT, RFE), this transformation was applied to the reduced feature set, while it was applied to the full feature set for other models. Convex regularized linear models were also evaluated post-feature selection for comparative purposes.

In contrast, preliminary analysis of dataset B revealed a nearly linear relationship between the target and the features, suggesting no significant benefit from feature transformation. Therefore, polynomial transformation was not applied to this dataset.

The regularization strength for each model, denoted by the parameter α, was determined via a 5-fold cross-validation across a grid of 100 logarithmically spaced values within the range of $(0.01, 100)$, optimizing for the maximum R^2 score.

For the evaluation of models, a total of 5 train-test splits were employed, with an 80%-20% ratio, generated by varying random states. This approach ensures that each statistical prediction metric reflects the mean of 5 independent evaluations on different training and testing splits, thereby mitigating biased estimation of prediction errors. The results are reported in terms of normalized nRMSE, the coefficient of determination (R^2), Learning Curves, the Area Under the nRMSE Learning Curve (AUC-nRMSE), Guardband G, and Compression Rate.

The AUC-nRMSE quantifies the generalization capability of a model, where a lower AUC-nRMSE value signifies superior generalization performance with fewer labeled samples. Specifically, AUC-nRMSE values were calculated for the initial 20 points of the learning curves to assess the models' performance when only a limited number of samples are available (600 labeled samples for dataset A and

100 for dataset B).

The Compression Rate (C) measures the relative reduction in data representation size achieved by a compression algorithm. It is defined as the ratio of the uncompressed size to the compressed size. In this context, C is computed by dividing the total number of ROs (or, in the case of prior Polynomial expansion, the number of computed coefficients) by the number of non-zero coefficients in the linear model ($C = \frac{totInput}{Coeff \neq 0}$). A higher C value indicates a smaller model size, reflecting fewer non-zero coefficients and reduced information retention requirements for logging.

Experiments were conducted on a server equipped with an Intel ® Core™ i9-9900K CPU @ 3.60GHz with 16 cores and 32 GiB of RAM. The models were implemented in *Python* using established machine learning libraries. Scikit-learn [105] was used for standard linear regression and convex regularization methods (LASSO, Elastic Net, Ridge, OMP). For models employing non-convex penalties, the scikit-glm [33] package was utilized, an extension of scikit-learn designed for generalized linear models with custom penalty terms.

7.5.2 Multicollinearity analysis

Feature sets obtained through CorrT filtering, Recursive Feature Elimination (RFE), and various regularized linear models (including LASSO, ElasticNet, Orthogonal Matching Pursuit (OMP), along with LGP and MCP for non-convex regularization) were evaluated in comparison to the full feature set. The objective was to assess the efficacy of each method in mitigating multicollinearity during feature selection. A Random Forest was employed as the baseline estimator for this evaluation, with training performed on 5 different train-test splits (80%-20%).

The results, presented in table 7.1, demonstrate that Random Forest exhibits strong performance across both datasets. Although each feature selection technique resulted in varying numbers of selected features, there is no substantial difference in overall model performance among the methods.

The analysis reveals that all feature sets generally yield comparable R^2 scores. For both datasets, reducing the number of SMONs does not markedly affect model accuracy. Nonetheless, some feature selection approaches prove more effective at reducing feature dimensionality while preserving performance. Notably, the CorrT method selects only 5 features for Dataset A while maintaining a relatively high R^2 score. For Dataset B, a more compact feature set helps slightly mitigate model overfitting, resulting in a marginal enhancement in prediction accuracy, as observed with LASSO and ElasticNet.

The presence of correlation among features within the datasets indicates that not all features contribute equally to the model's performance. This disparity is illuminated through the computation of permutation importance scores, a model

Table 7.1: R^2 Score (%) on datasets A and B with RandomForest and different feature sets (5-splits)

Method	Dataset A: #Features	Dataset A: R^2 Score	Dataset B: #Features	Dataset B: R^2 Score
All	27	97.94 ± 0.10	579	93.53 ± 1.02
RFE	23	97.92 ± 0.11	359	93.48 ± 1.00
CorrT	5	97.32 ± 0.08	10	93.57 ± 0.85
OMP	11	97.87 ± 0.10	12	93.48 ± 1.19
LASSO	11	97.90 ± 0.12	50	93.98 ± 0.94
ElasticNet	12	97.87 ± 0.13	83	93.70 ± 0.89
MCP	9	97.62 ± 0.08	16	92.86 ± 0.75
LGP	10	97.81 ± 0.13	24	93.27 ± 0.72

inspection technique that evaluates the impact of shuffling individual feature values on model accuracy [3]. The permutation importance scores for the features of the two datasets are illustrated in figs. 7.3 and 7.4.

Permutation importance was determined by averaging the results over 10 permutations for each feature, with only those features causing a decrease in the R^2 score exceeding 0.1% being displayed.

For Dataset A, the analysis reveals that, aside from a few SMONs with considerable impact, most features exhibit negligible importance. This observation is somewhat paradoxical given the high test accuracy observed, suggesting that the model's performance is less sensitive to these less important features. In Dataset B, the extensive number of features and their high inter-correlation exacerbate the low importance values, with a maximum importance of only 1.5%. Negative permutation importance values indicate that the model may perform better on randomly shuffled data, implying that such features have minimal impact on predictive outcomes. This phenomenon, often seen in smaller datasets like Dataset B with approximately 400 samples, suggests that any observed accuracy improvement with shuffled data is likely due to random chance.

The CorrT method proves particularly effective in addressing multicollinearity by removing highly correlated features. This results in a noticeable increase in feature importance. In contrast, RFE does not significantly alleviate multicollinearity, as the feature importance for the reduced feature set remains similar to that of the full feature set. The RFE procedure did not substantially reduce the number of features (from 27 to 23 for Dataset A and from 579 to 359 for Dataset

B).

Regularized linear models demonstrate a greater capability to identify features with higher predictive value. For instance, in Dataset A, the importance of the second most impactful SMON (SMON 10) increases from 5-6% in the full and RFE feature sets to 10-11% with regularized linear models, effectively doubling its importance. Similarly, in Dataset B, the importance of each SMON more than doubles, with OMP notably selecting a minimal number of features, all of which exhibit significant importance.

This analysis indicates the following key points:

- Preprocessing steps for feature selection do not substantially enhance prediction accuracy or the identification of critical features. Regularized linear models are capable of achieving comparable prediction accuracy while also pinpointing important features.
- The CorrT method facilitates a significant reduction in the number of features, albeit with a minor reduction in prediction accuracy. This reduction is also achievable with the OMP technique, which selects features most correlated with the residuals, and can potentially be attained with other regularized linear models by adjusting the α parameter.
- The RFE technique is ineffective both in significantly reducing the number of features and in improving accuracy compared to the baseline.

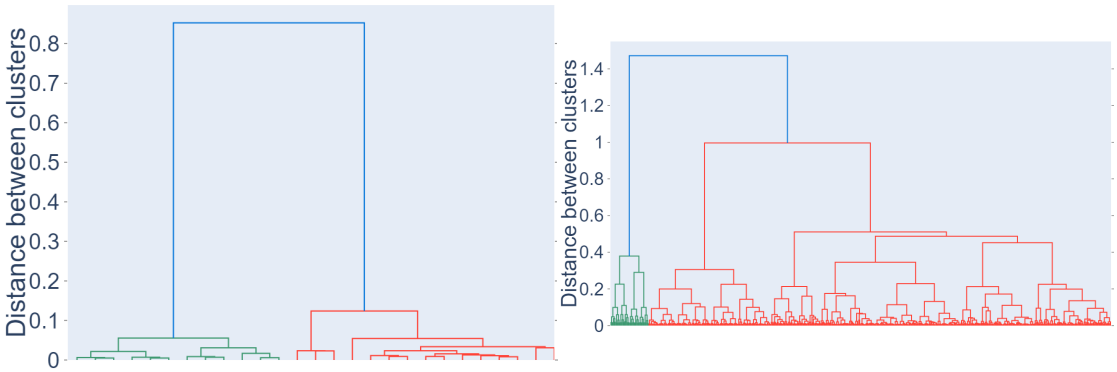


Figure 7.2: The Dendograms, results for the hierarchical clustering procedure, for dataset A (upper) and B (lower). By visually inspecting the dendograms, it is possible to set a threshold, to cut it at a certain level, extracting a certain number of cluster. On the y-axis, the distances between clusters. On the x-axis, the SMONs that form each (colored) cluster

7.5.3 Model metrics: Dataset A

Applying a degree-2 polynomial transformation to the various feature sets yields the following results:

- A total of 405 features are derived from 27 SMONs without prior FS.
- A total of 299 features are derived from 23 SMONs when using RFE.
- A total of 20 features are derived from 5 SMONs with the CorrT technique.

For dataset A, the prediction error across most algorithms converges to a similar level (fig. 7.5, table 7.2), with variations only in the number of coefficients utilized.

Regularization helps mitigate overfitting. This is evidenced by the upper plot in fig. 7.5, which depicts Polynomial Linear Regression. As the number of training samples approaches the number of features (approximately 400), the model achieves a perfect fit but fails to generalize to unseen data, resulting in an nRMSE of approximately 15% on test sets. This issue diminishes as the sample size increases, a behavior not observed with regularized models.

OMP typically achieves the highest compression rate, reducing the feature set from the full set to approximately 10:1, with minimal loss in accuracy (0.01 reduction in R^2). This provides significant savings in memory and computational resources. Among convex regularization methods, LASSO demonstrates the best balance between feature retention and prediction accuracy.

In terms of non-convex regularization, the LGP performs exceptionally well: it aligns nRMSE with that of using all features (1.66%) while achieving a higher compression rate of approximately 13:1.

Features selected by the CorrT technique exhibit the highest prediction errors. The removal of multicollinearity likely also excluded informative SMONs. If the resulting accuracy decrease is acceptable, CorrT remains a viable option due to its high compression rate (up to about 29:1 with OMP).

For this dataset, RFE procedures provide marginally better performance compared to no prior pruning (Ridge with RFE shows an R^2 score of 97.85%, whereas LASSO without RFE scores 97.84%). Despite this, incorporating an additional FS step is advantageous (ElasticNet/LASSO with RFE achieves an R^2 of 97.86%, compared to Ridge's 97.85%). However, this benefit comes with the cost of increased computational effort required to obtain the reduced feature set (23 SMONs).

The $L_{0.5}$ penalty does not perform well, with performance highly dependent on the training set, suggesting potential overfitting (evident from the shark-tooth pattern in the learning curve, fig. 7.5).

To assess the correlation between generalization capability and training sample size, the nRMSE-AUC is computed (table 7.2). Regularized models exhibit the

lowest AUC, indicating they require fewer samples to achieve effective training. Simpler models, in terms of fewer parameters or lower complexity, generally generalize better with fewer samples, reflecting the *bias-variance tradeoff* [65]. LASSO achieves the lowest AUC-nRMSE, representing the best compromise between the number of training samples and nRMSE.

The analysis indicates that incorporating a prior FS step does not substantially enhance prediction performance. In practice, allowing the models to select features autonomously yields satisfactory accuracy with reduced effort.

Regarding the selection of SMONs, linear models provide high interpretability. By examining the model coefficients, it is possible to determine which SMONs contribute to the selected polynomial terms. Consequently, any original SMONs that do not appear among the selected terms can be excluded. Non-convex regularized linear models facilitate the pruning of a number of SMONs without compromising accuracy. For instance, the LGP can effectively reduce the feature set by eliminating 6 SMONs while maintaining performance.

7.5.4 Model metrics: Dataset B

In the absence of polynomial transformation for this dataset, the number of coefficients directly reflects the number of SMONs selected. Ridge Regression, without feature pruning, yields the highest overall accuracy, indicating that more SMONs generally provide more information about the device's performance. However, the goal is to balance prediction accuracy with the number of SMONs.

Starting with the full feature set, LASSO demonstrates its capability to retain the most informative SMONs while achieving a compression rate of 9.65:1, with only a slight decrease in prediction accuracy (nRMSE increases from 0.86% to 0.89%). On the other hand, the $L_{0.5}$ penalty model continues to exhibit a shark-tooth pattern, reflecting suboptimal performance.

The LGP regularization further reduces the feature set, albeit with a slight increase in prediction accuracy. For the CorrT feature set, all models converge to similar performance levels, with R^2 values ranging from 94.31% (achieved by OMP) to 94.22% (achieved by non-regularized Linear Regression). The CorrT method enables a substantial reduction in the number of coefficients (to 8 with CorrT + OMP), but results in the lowest prediction accuracy, though nRMSE remains below 1%.

The RFE method did not significantly prune the feature set but managed to maintain reasonable prediction accuracy. In terms of the tradeoff between accuracy and compression rate, LASSO emerges as the most effective model, selecting approximately 60 ROs with an nRMSE of 0.89%, regardless of whether prior feature selection is used or not (refer to table 7.3). Additionally, LASSO achieves a favorable AUC value.

Table 7.2: Error metrics on A, 5-folds average, divided by feature set (in bold)

Model (per SMONs set)	Selected Coef (over Total)	Selected SMONs (over Total)	C	nRMSE%	R^2%	G%	AUC
All SMONs							
Lin. R.	405/405	27/27	1:1	1.75	97.58%	11.16	28.62
Ridge	405/405	27/27	1:1	1.66	97.84%	10.53	10.14
LASSO	55/405	26/27	7.36:1	1.66	97.83%	10.55	9.98
ElasticNet	76/405	26/27	5.32:1	1.66	97.84%	10.54	10.15
OMP	41/405	23/27	9.87:1	1.67	97.81%	10.61	10.45
All SMONs (Non convex)							
$\ell_{\frac{1}{2}}$	68/405	26/27	5.95:1	1.68	97.76%	10.72	16.88
LGP	31/405	21/27	13.06:1	1.66	97.83%	10.57	10.26
MCP	31/405	22/27	13.06:1	1.71	97.69%	10.90	10.50
SCAD	29/405	20/27	13:96	1.71	97.67%	10.92	10.57
CorrT							
Lin. R.	20/405	5/27	20.25:1	1.88	97.22	11.96	11.53
Ridge	20/405	5/27	20.25:1	1.88	97.22	11.98	11.31
LASSO	17/405	5/27	23.82:1	1.88	97.22	11.98	11.32
ElasticNet	19/405	5/27	21.32:1	1.88	97.22	12.35	11.29
OMP	14/405	5/27	28.93:1	1.88	97.22	12.33	11.41
RFE.							
Lin. R.	299/405	23/27	1.35:1	1.71	97.69	10.90	25.60
Ridge	299/405	23/27	1.35:1	1.65	97.85	10.50	10.09
LASSO	67/405	23/27	6.04:1	1.65	97.86	10.49	10.03
ElasticNet	78/405	23/27	5.19:1	1.65	97.86	10.49	10.01
OMP	42/405	22/27	9.64:1	1.66	97.83	10.56	10.44

Employing a prior feature selection step does not provide significant gains in prediction accuracy. While the CorrT technique allows for the smallest feature set with acceptable prediction performance, similar results can be achieved with OMP without prior feature selection, which also offers better prediction performance compared to CorrT. Non-convex regularization methods generally performed worse in terms of prediction accuracy but provided higher compression rates.

7.5.5 Time Analysis

Filtering methods are generally known for their computational efficiency, typically exhibiting linear or near-linear time complexity relative to the number of features. This is due to their evaluation of each feature independently. In the context of this framework, extracting feature sets using the CorrT technique was completed in a matter of seconds.

Table 7.3: Error metrics on B, 5-folds average, divided by feature set (in bold)

Model	Coef No. (over Input)	C	nRMSE%	R^2%	G%	AUC
All SMONs						
Lin. R.	579/579	1:1	1.15	92.26	9.29	0.76
Ridge	579/579	1:1	0.86	95.64	6.94	0.74
LASSO	60/579	9.65:1	0.89	95.31	7.17	0.76
E.Net	82/579	7.06:1	0.92	94.97	7.42	0.75
OMP	18/579	32.16:1	0.93	94.78	7.59	0.89
All SMONs (Non-Convex)						
$\ell_{\frac{1}{2}}$	37/579	15.65:1	1.13%	92.52%	9.14	1.17
LGP	25/579	23.16:1	0.93%	94.91%	7.51	0.89
MCP	19/579	30.47:1	0.96%	94.57%	7.69	0.86
SCAD	14/579	41.36:1	0.97%	94.39%	7.81	0.87
CorrT						
Lin. R.	10/10	1:1	0.99	94.22	7.96	1.24
Ridge	10/10	1:1	0.98	94.25	7.94	0.80
LASSO	8/10	1.25:1	0.98	94.29	7.91	0.78
Elastic	9/10	1.11:1	0.98	94.27	7.92	0.77
OMP	8/10	1.25:1	0.98	94.31	7.90	0.80
RFE.						
Lin. R.	358/358	1:1	3.72	17.15	30.90	0.76
Ridge	358/358	1:1	0.87	95.47	7.05	0.73
LASSO	75/358	4.77	0.89	95.28	7.20	0.77
E.Net	71/358	1.15	0.92	94.97	7.43	0.74
OMP	16/358	22.37	0.93	94.83	7.51	0.88

In contrast, the time complexity of RFE is influenced by the choice of the underlying model. RFE iteratively fits the model and assesses feature importance, resulting in a linear time complexity relative to the number of features, augmented by the training complexity of the model used. For the current framework, extracting feature sets via RFE required several hours.

CS and regularized linear models incorporate feature selection into the model training process. Therefore, their time complexity is primarily determined by the training complexity of the underlying model, without adding significant overhead

to the training procedure. Nevertheless, training models such as LASSO or ElasticNet is computationally more intensive compared to Ridge or Linear Regression, with the former requiring several minutes versus a few seconds for the latter.

Non-convex regularization models are comparatively faster to train than their convex counterparts, owing to more efficient underlying optimization algorithms.

Each method presents a tradeoff between the quality of the feature sets extracted and the time required for extraction. However, empirical results indicate that regularized linear models consistently achieve a favorable balance between prediction accuracy and compression rate. Their efficiency in training further solidifies their position as the preferred choice in this framework.

7.6 Conclusions

In this chapter, the integration of Compressed Sensing and regularized linear models has been proposed as a cohesive framework for addressing MCU performance screening. A comparative analysis of three primary feature selection techniques revealed that preliminary feature selection is not essential in this context. Instead, linear models with regularization effectively extract critical information about the optimal SMONs for predicting $F_{\max}$, which is crucial for determining which SMONs should be included in the final product.

Specifically, wrapper methods do not substantially enhance prediction accuracy or reduce the number of features significantly while incurring high computational costs. Filter methods, including the proposed dendrogram approach, can efficiently reduce the feature set. However, their reliance solely on feature information may lead to the exclusion of important features necessary for subsequent supervised regression tasks, potentially diminishing prediction accuracy compared to embedded feature selection methods.

Ultimately, regularized linear models present an optimal balance between prediction accuracy and the number of retained SMONs, making them the preferred choice for datasets A and B, particularly LASSO and LGP. OMP achieves the highest compression rate while preserving reasonable accuracy. Non-convex regularization techniques, notably LGP, offer a middle ground between LASSO and OMP in terms of feature retention and prediction accuracy.

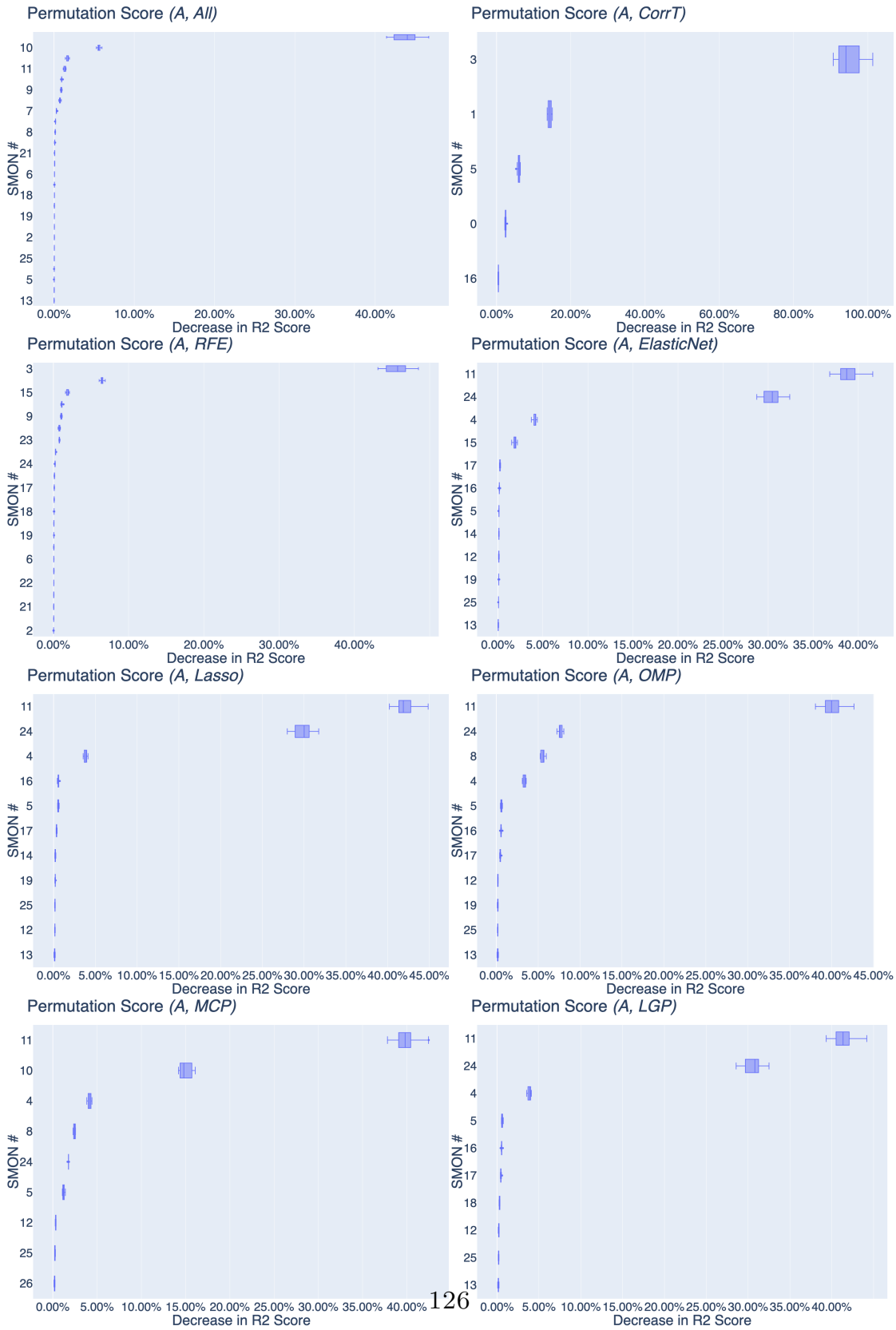


Figure 7.3: Permutation Importance Index for dataset A

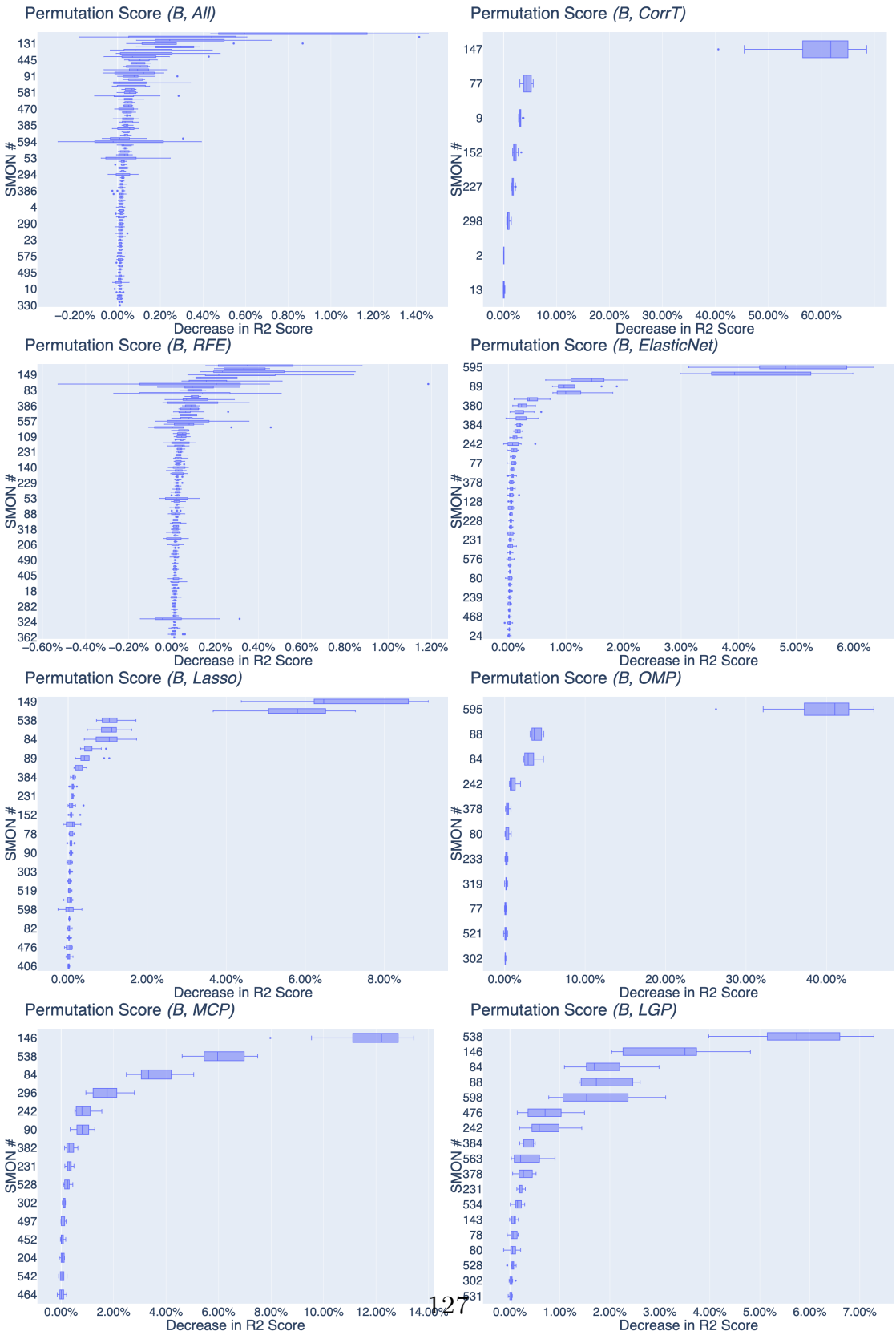


Figure 7.4: Permutation Importance Index for dataset B

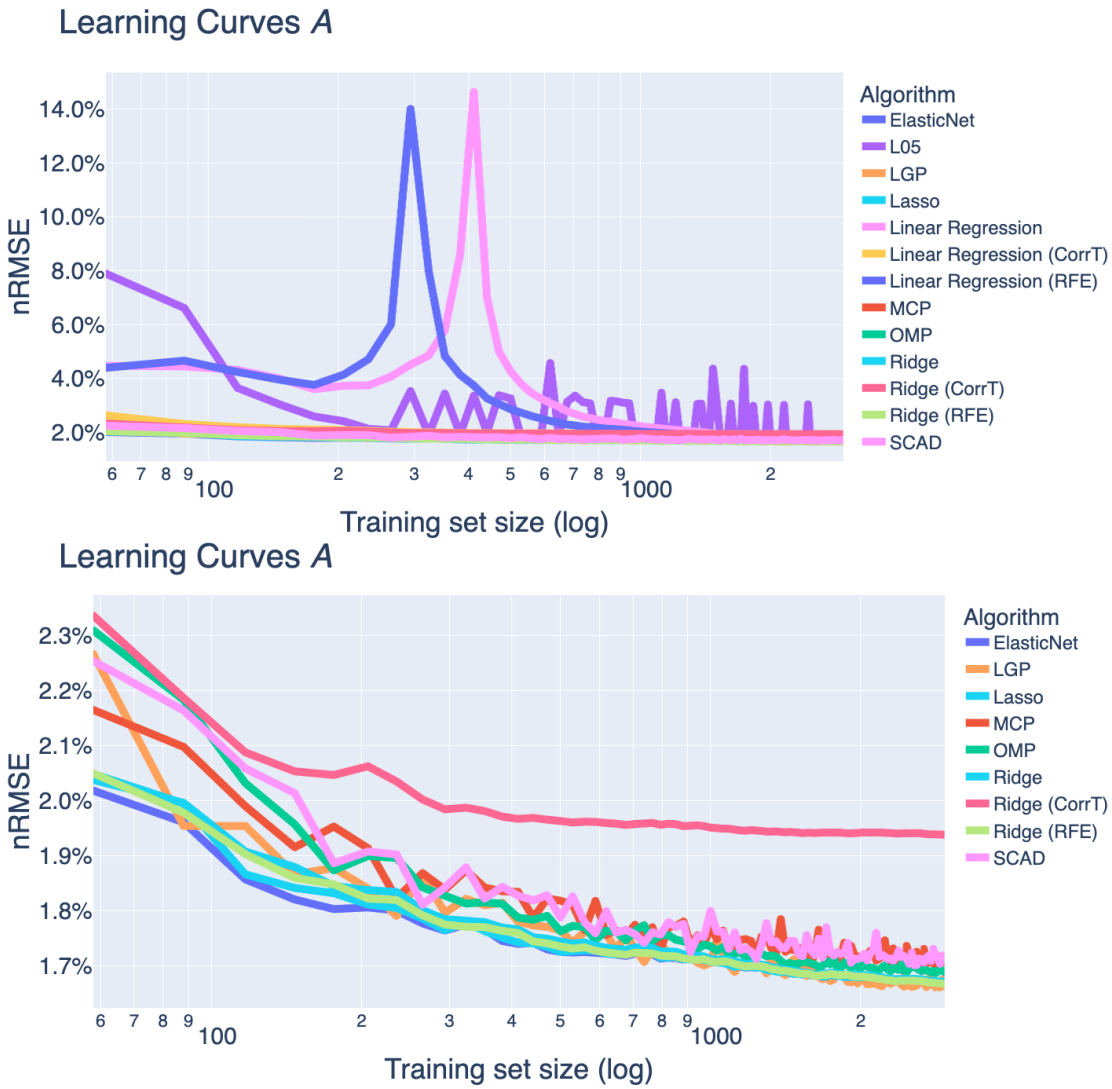


Figure 7.5: Learning curves for different models on A dataset. The x-axes is the number of training samples (log scale) while the y-axes is the nRMSE computed on the test set. The upper plot shows the overfitting of Linear Regression and the Shark-tooth behavior of $\ell_{\frac{1}{2}}$. We disabled that traces in the lower plot. CorrT techniques collapse to 1.94% nRMSE. All the other models are aligned in terms of final error metrics, with slightly better results in terms of error/coefficient for the non-convex LGP model.

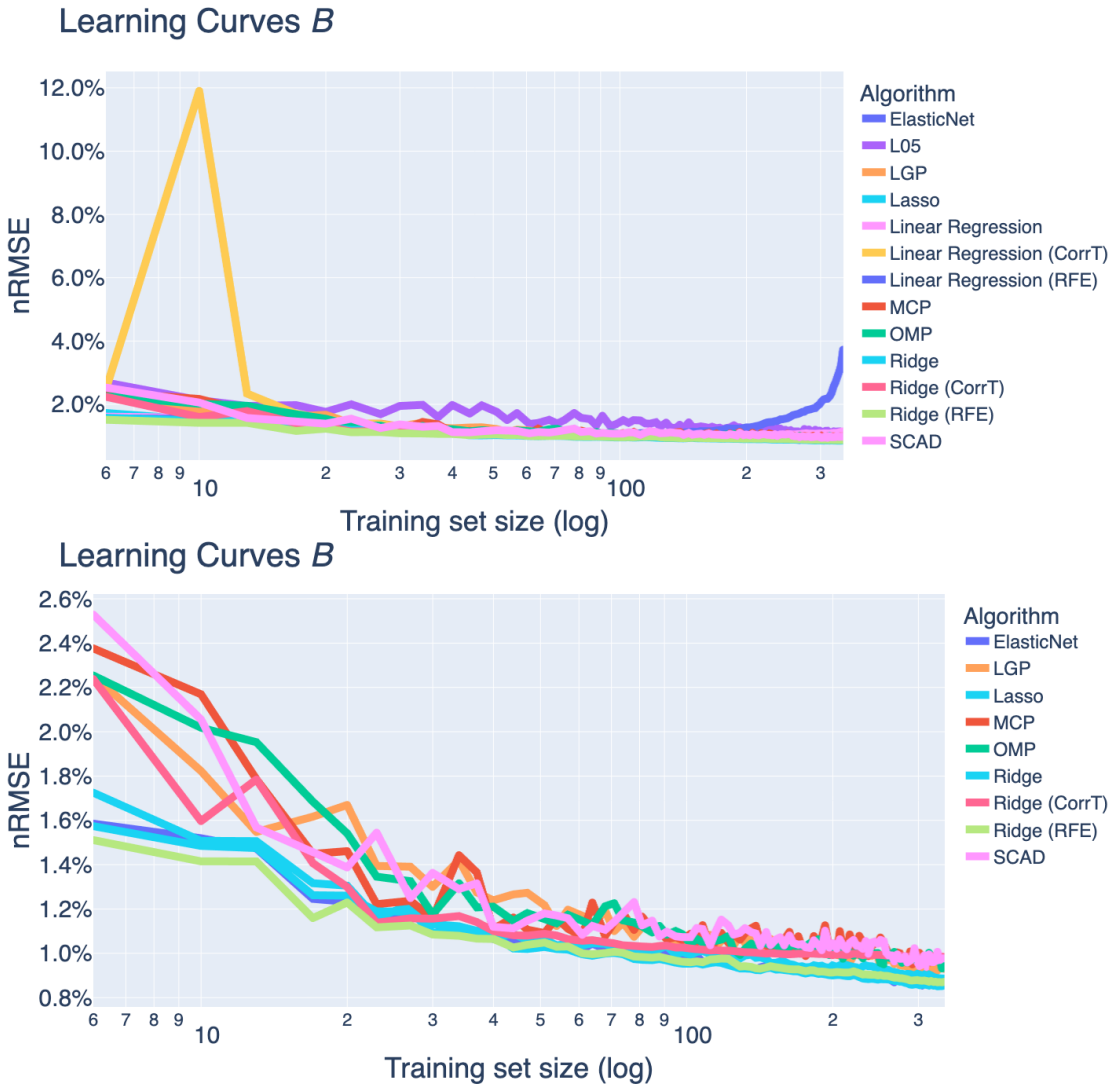


Figure 7.6: Learning curves for different models on dataset B. The x-axes is the number of training samples (log scale) while the y-axes is the nRMSE computed on the test set. The upper plot shows, even for this dataset, the overfitting of Linear Regression and the Shark-tooth behavior of $\ell_{\frac{1}{2}}$. We disabled the traces in the lower plot. LASSO model is the most able to reach good balance between number of feature chosen (60) and prediction accuracy (0.89% nRMSE). Non-convex techniques tend to prefer sparser solutions, at the cost of an increase in the prediction error.

Chapter 8

Conclusion

This work consolidated multiple methodologies aimed at developing and enhancing an advanced ML-based framework for the prediction accuracy of MCUs maximum operating frequency, focusing on various aspects such as multi-label frameworks, deep learning, and feature selection using compressed sensing. Through the integration of these techniques, the study addresses critical challenges in MCU performance prediction during production screening, offering a comprehensive approach to model development and optimization.

The research advances existing MCU performance prediction models by incorporating feature transformation techniques and multi-label learning, resulting in reduced prediction error and minimized guardband, thereby improving overall process yield. Specifically, multi-label regression facilitated a more nuanced analysis of interdependencies among testing frequencies target variables, significantly lowering the prediction error for the critical patterns, $P_{\min}$. This multi-label approach proved particularly effective when several highly correlated labels are present, ensuring more accurate and reliable predictions.

In parallel, the application of dimensionality reduction and regularized linear models allowed for a significant reduction in the number of features considered by the predictive models, streamlining the modeling process. The use of polynomial transformations further enhanced model performance by creating non-linear features that better captured the underlying relationships in the data. These enhancements enabled the use of simpler, more interpretable models, which are preferable in many industrial applications due to their transparency and ease of implementation.

The study also demonstrated the effectiveness of active learning (AL) techniques in reducing the number of labeled samples required to achieve a desired level of prediction accuracy. By employing strategies such as Local Outlier Factor and Hausdorff Distance sampling, the training set size was significantly reduced compared to random sampling methods, leading to substantial time savings in

dataset creation. The combined use of AL and multi-label regression resulted in more robust and efficient ML models, further decreasing prediction errors and improving process yield.

Furthermore, the application of deep learning and transfer learning techniques within this framework showed promising results, particularly in the context of inter-family knowledge transfer. Deep neural networks trained on one dataset were successfully fine-tuned and adapted to new, related datasets, illustrating the feasibility of applying pre-trained models across different product families within the same technological domain. This approach not only enhanced prediction accuracy but also demonstrated the generalization capabilities of the models, making them highly applicable in diverse industrial scenarios.

Finally, the exploration of compressed sensing and regularized linear models provided a unifying framework for feature selection, particularly in the context of SMON $F_{\max}$ prediction. The findings revealed that prior feature selection might be unnecessary, as linear models with regularization effectively extract relevant information. Among the methods examined, regularized linear models, particularly LASSO and LGP, emerged as optimal choices for balancing prediction accuracy and feature retention.

8.1 Future Work

Future research directions should aim at expanding the applicability and enhancing the efficiency of the proposed framework for performance screening and device testing. Key areas for further investigation include:

- **Enhanced Integration of Active Learning and Transfer Learning:** The combination of AL and transfer learning techniques should be further explored to refine the framework's adaptability across various MCU families and semiconductor technologies. Developing new sampling strategies and transfer learning paradigms tailored to specific domains or data scarcity scenarios could result in more efficient model training, reducing labeling costs and improving generalization to unseen data.
- **Application to Real-Time Production Environments:** Extending the proposed methods to real-time performance screening applications is another promising avenue. Implementing this framework in live production environments would require optimizing for speed and scalability. Future studies could investigate the use of distributed computing techniques, parallel processing, and real-time data pipelines to handle the large-scale data typical in semiconductor manufacturing.

- **Incorporation of Non-Convex Regularization with Deep Learning Models:** The integration of non-convex regularization techniques, such as LGP, with deep learning models warrants further exploration. This could help leverage the strengths of both approaches, combining the interpretability and feature selection benefits of regularized models with the powerful learning capacity of neural networks, potentially leading to further gains in accuracy and efficiency.
- **Development of Hybrid Models for Improved Prediction Accuracy:** Building on the findings, future work could focus on creating hybrid models that combine the strengths of various regularization techniques (both convex and non-convex) with advanced machine learning architectures, such as ensemble learning or meta-learning frameworks. These models could provide a more balanced trade-off between feature retention, prediction accuracy, and computational efficiency.
- **Extension to Broader Device Testing Applications:** The proposed framework's application could be extended beyond MCU performance screening to other aspects of semiconductor device testing, such as yield optimization, fault diagnosis, and lifetime prediction. Investigating its effectiveness in these areas could provide new insights into optimizing the overall manufacturing process, enhancing reliability and reducing costs.
- **Advanced Interpretability Techniques for Industrial Application:** Future research should explore advanced interpretability techniques, such as SHAP (SHapley Additive exPlanations) [93, 94] values or counterfactual explanations, to better understand the decision-making process of ML models within the proposed framework. This could increase trust and acceptance among stakeholders, facilitating deployment in industrial contexts where transparency and accountability are crucial.
- **Investigation of Feature Selection under Varying Environmental Conditions:** To further enhance robustness, it is essential to study the impact of varying environmental conditions, such as temperature and voltage fluctuations, on feature selection and model performance. Developing methods that account for such variations could lead to more resilient prediction models that maintain accuracy across a broader range of operating conditions.

Overall, the proposed framework holds significant potential for broader application within the context of performance screening, device testing, and semiconductor manufacturing. Future work should continue to refine these methodologies,

enhancing their applicability and impact across different domains, ultimately driving improvements in production quality, efficiency, and reliability.

8.1.1 Side Activities

The techniques and knowledge acquired during these years have allowed for an extension of research activities beyond the initial focus on MCU performance screening. In particular, expertise in Deep Learning has been applied to the study of Neural Network reliability, resulting in several collaborative research efforts with colleagues from École Centrale de Lyon [18, 17, 19].

Furthermore, experience gained in data analysis has been leveraged in a different context, specifically for the analysis of mass spectra of exhaled breath [23, 22]. This work was conducted in collaboration with NanoTech Analysis S.r.l. (NTA) [1]. The acquired mass spectra from the exhaled breath of hospitalized patients were utilized to discriminate between COVID-19 positive and negative cases. Filtering techniques and mass-spectra preprocessing procedures were developed, with findings used to define future research directions in the field of mass spectrometry analysis. Finally, collaboration with colleagues from the School of Micro- and Nano-Electronics at Zhejiang University, Hangzhou, China, and Delft University of Technology, Netherlands, has led to advancements in research on FeFET technologies.

[1] https://nanotechanalysis.com/

Appendix A

List of Published Papers

During these three years of my PhD, in synergy with my research team and colleagues, we conducted several research activities that resulted in multiple publications. The primary focus was on MCU performance screening, although additional research was also pursued in areas such as neural network reliability studies and breath analysis. In total, the following have been published:

- 2 posters
- 10 conference papers
- 4 journal papers
- 1 arXiv paper

As of today (December 2, 2024), the following have been submitted for review:

- 4 conference papers

Below is a comprehensive list of the papers that have been published or submitted for review, divided by topic:

Microcontroller Performance Screening

- N. Bellarmino et al., *Exploiting Active Learning for Microcontroller Performance Prediction*, 2021 IEEE European Test Symposium (ETS), Bruges, Belgium, 2021 [27]
- F. Angione et al., *Test, Reliability and Functional Safety Trends for Automotive System-on-Chip* 2022 IEEE European Test Symposium (ETS), Barcelona, Spain, 2022 [8]

- N. Bellarmino, R. Cantoro, M. Huch, T. Kilian, U. Schlichtmann and G. Squillero, *Microcontroller Performance Screening: Optimizing the Characterization in the Presence of Anomalous and Noisy Data*, 2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS), Torino, Italy, 2022 [31]

- N. Bellarmino, R. Cantoro, M. Huch, T. Kilian, U. Schlichtmann and G. Squillero, *Semi-Supervised Deep Learning for Microcontroller Performance Screening*, 2023 IEEE European Test Symposium (ETS), Venezia, Italy, 2023 [29]

- N. Bellarmino et al., *A Multilabel Active Learning Framework for Microcontroller Performance Screening*, in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 10, pp. 3436-3449, Oct. 2023 [20]

- N. Bellarmino, R. Cantoro, M. Huch, T. Kilian, U. Schlichtmann and G. Squillero, *Enabling Inter-Product Transfer Learning on MCU Performance Screening*, 2023 IEEE 32nd Asian Test Symposium (ATS), Beijing, China, 2023 [26]

- N. Bellarmino, R. Cantoro, M. Huch, T. Kilian, U. Schlichtmann and G. Squillero, *Feature Selection for Cost Reduction In MCU Performance Screening*, 2023 IEEE 24th Latin American Test Symposium (LATS), Veracruz, Mexico, 2023 [28]

- N. Bellarmino, R. Cantoro, M. Huch, T. Kilian, U. Schlichtmann and G. Squillero, *Embedded Feature Selection in MCU Performance Screening*, IEEE 2nd International conference on Design, Test & Technology of Integrated Systems (DTTIS), Aix-en-Provence, France, 2024 [25]

- N. Bellarmino, R. Cantoro, M. Huch, T. Kilian, U. Schlichtmann and G. Squillero, *Deep Learning Strategies for Labeling and Accuracy Optimization in Microcontroller Performance Screening*, in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems [24]

- N. Bellarmino, R. Cantoro, M. Huch, T. Kilian, U. Schlichtmann and G. Squillero, *COSMO: COmpressed Sensing for Models and logging Optimization in MCU Performance Screening*, in IEEE Transactions Computers [21]

- Bellarmino, N., R. Cantoro, M. Huch, T. Kilian, U. Schlichtmann and G. Squillero, *Transfer learning in MCU performance screening,* Poster@ ITC 2023, Available at: `https://iris.polito.it/handle/11583/2981875` [16]

- Bellarmino, N., R. Cantoro, M. Huch, T. Kilian, U. Schlichtmann and G. Squillero *Machine Learning for Microcontroller Performance Screening*, Poster@ETS23, Available at: https://iris.polito.it/handle/11583/2981874 [15]

Neural Network Reliability

- N. Bellarmino, A. Bosio, R. Cantoro, A. Ruospo, E. Sanchez, G. Squillero, *Investigating on Gradient Regularization for Testing Neural Networks*. LOD 2024: 10th International Conference on machine Learning, Optimization and Data science, Sep 2024, Riva de Sole, Italy. [18]

- N. Bellarmino, A. Bosio, R. Cantoro, A. Ruospo, E. Sanchez, G. Squillero, *DEAR-CNN: Data-Efficient Assessment of Resiliency in Convolutional Neural Networks*. Submitted for Review. [17]

- N. Bellarmino, A. Bosio, R. Cantoro, G. Squillero et Al., *LOL: Learning tO Learn Training Dynamics for CNN Reliability*. Submitted for Review. [19]

Computing-in-Memory (CiM) and FeFET devices

- Changhao Wang, Nicolo' Bellarmino et Al., *EAR: Endurance-Aware Retraining for Efficient DNN Inference on FeFET-Based Accelerators*, Submitted for Review.

- Changhao Wang, Nicolo' Bellarmino et Al., *A Data-Driven Machine Learning-Based Model Methodology for Ferroelectric Memory*, Submitted for Review.

- Sicong Yuan, Changhao Wang, Moritz Fieback, Hanzhi Xun, Mottaqiallah Taouil, Li Xiuyan, Lin Wang, Danyang Chen, Nicolò Bellarmino, Riccardo Cantoro, Said Hamdioui, *Device-Aware Test for Anomalous Charge Trapping in FeFETs*, in 30th Asia and South Pacific Design Automation Conference *ASP-DAC* 2025

ML in Biochemistry

- Bellarmino N, Bozzini G, Cantoro R, et al. *COVID-19 Detection from Mass Spectra of Exhaled Breath.* arXiv; 2023 [23]

- Bellarmino N, Bozzini G, Cantoro R, et al. *COVID-19 Detection from Exhaled Breath*, in *Scientific Report*; 2024 [22]

Pure ML

- Bellarmino, N., Cantoro, R., Squillero, G. (2024). *U-FLEX: Unsupervised Feature Learning with Evolutionary eXploration.* In: Nicosia, G., Ojha, V., La Malfa, E., La Malfa, G., Pardalos, P.M., Umeton, R. (eds) Machine Learning, Optimization, and Data Science. LOD 2023. Lecture Notes in Computer Science, vol 14505. Springer, Cham [30]

Bibliography

[1] Abien Fred Agarap. "Deep Learning using Rectified Linear Units (ReLU)". In: *arXiv preprint arXiv:1803.08375* (2018). URL: https://arxiv.org/abs/1803.08375.

[2] Hani Almousli and Pascal Vincent. "Semi Supervised Autoencoders: Better Focusing Model Capacity during Feature Extraction". In: *Neural Information Processing*. Ed. by Minho Lee et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 328–335. ISBN: 978-3-642-42054-2.

[3] André Altmann et al. "Permutation importance: a corrected feature importance measure". In: *Bioinformatics* 26.10 (Apr. 2010), pp. 1340–1347. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btq134. eprint: https://academic.oup.com/bioinformatics/article-pdf/26/10/1340/48851160/bioinformatics_26_10_1340.pdf. URL: %5Curl%7Bhttps://doi.org/10.1093/bioinformatics/btq134%7D.

[4] Laith Alzubaidi et al. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". In: *Journal of Big Data* 8.1 (Mar. 2021), p. 53. ISSN: 2196-1115. DOI: 10.1186/s40537-021-00444-8. URL: %5Curl%7Bhttps://doi.org/10.1186/s40537-021-00444-8%7D.

[5] Hussam Amrouch et al. "Special Session: Machine Learning for Semiconductor Test and Reliability". In: *2021 IEEE 39th VLSI Test Symposium (VTS)*. 2021, pp. 1–11. DOI: 10.1109/VTS50974.2021.9441052.

[6] Young-Jae An et al. "All-Digital ON-Chip Process Sensor Using Ratioed Inverter-Based Ring Oscillator". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.11 (2016), pp. 3232–3242. DOI: 10.1109/TVLSI.2016.2550603.

[7] Plamen Angelov. *Autonomous Learning Systems: From Data Streams to Knowledge in Real-time*. en. 1st ed. Wiley, Dec. 2012. ISBN: 9781119951520. DOI: 10.1002/9781118481769. URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9781118481769.

[8] F. Angione et al. “Test, Reliability and Functional Safety Trends for Automotive System-on-Chip”. In: *2022 IEEE European Test Symposium (ETS)*. 2022, pp. 1–10. DOI: 10.1109/ETS54262.2022.9810388.

[9] Dana Angluin. “Queries and Concept Learning”. In: *Mach. Learn.* 2.4 (Apr. 1988), pp. 319–342. ISSN: 0885-6125. DOI: 10.1023/A:1022821128753. URL: %5Curl%7Bhttps://doi.org/10.1023/A:1022821128753%7D.

[10] Shojiro Asai, ed. *VLSI Design and Test for Systems Dependability*. Springer Japan, 2019. DOI: 10.1007/978-4-431-56594-9.

[11] Mariette Awad and Rahul Khanna. “Support Vector Regression”. In: *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. Berkeley, CA: Apress, 2015, pp. 67–80. ISBN: 978-1-4302-5990-9. DOI: 10.1007/978-1-4302-5990-9_4. URL: https://doi.org/10.1007/978-1-4302-5990-9_4.

[12] D. Azzimonti et al. “Comparison of domain adaptation and active learning techniques for quality of transmission estimation with small-sized training datasets [Invited]”. In: *IEEE OSA* 13.1 (2021), A56–A66. DOI: 10.1364/JOCN.401918.

[13] Baosenguo. “1D-CNN: Kaggle-MoA 2nd Place Solution”. In: *Kaggle* (Dec. 2020). URL: %5Curl%7Bhttps://github.com/baosenguo/Kaggle-MoA-2nd-Place-Solution%7D.

[14] Jonathan T. Barron. “Continuously Differentiable Exponential Linear Units”. In: *arXiv* (2017). DOI: 10.48550/ARXIV.1704.07483. URL: %5Curl%7Bhttps://arxiv.org/abs/1704.07483%7D.

[15] N. Bellarmino et al. *Machine Learning for Microcontroller Performance Screening*. 2023. URL: https://iris.polito.it/handle/11583/2981874.

[16] N. Bellarmino et al. *Transfer Learning in MCU Performance Screening*. 2023. URL: https://iris.polito.it/handle/11583/2981875.

[17] Nicolo Bellarmino et al. *DEAR-CNN: Data-Efficient Assessment of Resiliency in Convolutional Neural Networks*. 2024.

[18] Nicolo Bellarmino et al. “Investigating on Gradient Regularization for Testing Neural Networks”. In: *LOD 2024 : 10th International Conference on machine Learning, Optimization and Data science*. Riva de Sole, Italy, Sept. 2024. URL: https://hal.science/hal-04604264.

[19] Nicolo Bellarmino et al. *LOL: Learning tO Learn Training Dynamics for CNN Reliability*. 2024.

[20] Nicolò Bellarmino et al. "A Multi-Label Active Learning Framework for Microcontroller Performance Screening". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2023), pp. 1–1. DOI: 10.1109/TCAD.2023.3245989.

[21] Nicolò Bellarmino et al. "COSMO: COmpressed Sensing for Models and logging Optimization in MCU Performance Screening". In: *IEEE Transactions on Computers* (2024), pp. 1–1.

[22] Nicolò Bellarmino et al. *COVID-19 Detection from Exhaled Breath*. 2023. URL: https://europepmc.org/article/PPR/PPR670245.

[23] Nicolò Bellarmino et al. *COVID-19 Detection from Mass Spectra of Exhaled Breath*. 2023. URL: https://europepmc.org/article/PPR/PPR670245.

[24] Nicolò Bellarmino et al. "Deep Learning Strategies for Labeling and Accuracy Optimization in Microcontroller Performance Screening". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024), pp. 1–1. DOI: 10.1109/TCAD.2024.3436542.

[25] Nicolò Bellarmino et al. "Embedded Feature Selection in MCU Performance Screening". In: *IEEE 2nd International conference on Design, Test and Technology of Integrated Systems (DTTIS)*. 2024, pp. 1–6.

[26] Nicolò Bellarmino et al. "Enabling Inter-Product Transfer Learning on MCU Performance Screening". In: *IEEE Asian Test Symposium (ATS)*. 2023, pp. 1–6.

[27] Nicolò Bellarmino et al. "Exploiting Active Learning for Microcontroller Performance Prediction". In: *IEEE European Test Symposium (ETS)*. 2021, pp. 1–4. DOI: 10.1109/ETS50041.2021.9465472.

[28] Nicolò Bellarmino et al. "Feature Selection for Cost Reduction In MCU Performance Screening". In: *IEEE 24th Latin American Test Symposium (LATS)*. 2023, pp. 1–6. DOI: 10.1109/LATS58125.2023.10154495.

[29] Nicolò Bellarmino et al. "Semi-Supervised Deep Learning for Microcontroller Performance Screening". In: *IEEE European Test Symposium (ETS)*. 2023, pp. 1–6. DOI: 10.1109/ETS56758.2023.10174083.

[30] Nicolo' Bellarmino, Riccardo Cantoro, and Giovanni Squillero. "U-FLEX: Unsupervised Feature Learning with Evolutionary eXploration". In: *Machine Learning, Optimization, and Data Science*. Ed. by Giuseppe Nicosia et al. Cham: Springer Nature Switzerland, 2024, pp. 364–378. ISBN: 978-3-031-53969-5.

[31] Nicolo' Bellarmino et al. "Microcontroller Performance Screening: Optimizing the Characterization in the Presence of Anomalous and Noisy Data". In: *IEEE International Symposium on On-Line Testing and Robust System (IOLTS)*. 2022, pp. 1–10.

[32] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. *Advances in Optimizing Recurrent Networks*. 2012. arXiv: 1212.0901 [cs.LG].

[33] Q. Bertrand et al. "Beyond L1: Faster and better sparse models with skglm". In: *NeurIPS*. 2022.

[34] Hanen Borchani et al. "A survey on multi-output regression". In: *WIREs Data Mining and Knowledge Discovery* 5.5 (2015), pp. 216–233. DOI: https://doi.org/10.1002/widm.1157. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1157. URL: %5Curl%7Bhttps://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1157%7D.

[35] Vadim Borisov et al. "Deep Neural Networks and Tabular Data: A Survey". In: *IEEE Transactions on Neural Networks and Learning Systems* PP (Dec. 2022), pp. 1–21. DOI: 10.1109/TNNLS.2022.3229161.

[36] Leon Bottou, Frank E. Curtis, and Jorge Nocedal. "Optimization Methods for Large-Scale Machine Learning". In: *Society for Industrial and Applied Mathematics (SIAM) Review* 60.2 (2018), pp. 223–311. DOI: 10.1137/16M1080173. eprint: https://doi.org/10.1137/16M1080173. URL: %5Curl%7B%20https://doi.org/10.1137/16M1080173%20%7D.

[37] Markus M. Breunig et al. "LOF: Identifying Density-Based Local Outliers". In: *ACM SIGMOD, Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. Dallas, Texas, USA: Association for Computing Machinery, 2000, pp. 93–104. ISBN: 1581132174. DOI: 10.1145/342009.335388. URL: https://doi.org/10.1145/342009.335388.

[38] JM Brick and G. Kalton. "Handling missing data in survey research". In: *Statistical Methods in Medical Research* 5.3 (1996). PMID: 8931194, pp. 215–238. DOI: 10.1177/096228029600500302. eprint: https://doi.org/10.1177/096228029600500302. URL: %5Curl%7B%20https://doi.org/10.1177/09622802960050302%7D.

[39] J.B. Brockman and S.W. Director. "Predictive subset testing: optimizing IC parametric performance testing for quality, cost, and yield". In: *"IEEE Transactions on Semiconductor Manufacturing"* 2.3 (1989), pp. 104–113. DOI: 10.1109/66.29679.

[40] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (Mar. 2016), pp. 3932–3937. ISSN: 1091-6490. DOI: 10.1073/pnas.1517384113. URL: %5Curl%7Bhttp://dx.doi.org/10.1073/pnas.1517384113%7D.

[41] Stef van Buuren and Karin Groothuis-Oudshoorn. "mice: Multivariate Imputation by Chained Equations in R". In: *Journal of Statistical Software* 45.3 (2011), pp. 1–67. DOI: 10.18637/jss.v045.i03. URL: %5Curl%7Bhttps://www.jstatsoft.org/index.php/jss/article/view/v045i03%7D.

[42] Murray Campbell, A. Joseph Hoane, and Feng- hsiung Hsu. "Deep Blue". en. In: *"Artificial Intelligence"* 134.1 (Jan. 2002), pp. 57–83. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(01)00129-1.

[43] Emmanuel J. Candès and Terence Tao. "Near-Optimal Signal Recovery From Random Projections: Universal Encoding Strategies?" In: *IEEE Transactions on Information Theory* 52.12 (2006), pp. 5406–5425. DOI: 10.1109/TIT.2006.885507.

[44] R. Cantoro et al. "Machine Learning based Performance Prediction of Microcontrollers using Speed Monitors". In: *IEEE International Test Conference (ITC)*. 2020.

[45] Vito Cerone, Sophie M. Fosson, and Diego Regruto. "Fast sparse optimization via adaptive shrinkage". In: *IFAC-PapersOnLine* 56.2 (2023), pp. 10390–10395.

[46] Tianfeng Chai and R.R. Draxler. "Root Mean Square Rrror (RMSE) or Mean Absolute Error (MAE)?– Arguments Against Avoiding RMSE in the Literature". In: *Geoscientific Model Development* 7 (June 2014), pp. 1247–1250. DOI: 10.5194/gmd-7-1247-2014.

[47] Tuck Boon Chan et al. "DDRO: A Novel Performance Monitoring Methodology Based on Design-Dependent Ring Oscillators". In: *Thirteenth International Symposium on Quality Electronic Design (ISQED)*. May 2012, pp. 633–640. DOI: 10.1109/ISQED.2012.6187559.

[48] C. Chang and C. H. Wen. "Accurate performance evaluation of VLSI designs with selected CMOS process parameters". In: *IET Circuits, Devices Systems* 12.1 (2018), pp. 116–123. ISSN: 1751-8598. DOI: 10.1049/iet-cds.2017.0097.

[49] O. Chapelle, B. Scholkopf, and A. Zien Eds. "Semi-Supervised Learning". In: *IEEE Transactions on Neural Networks* 20.3 (2009), pp. 542–542. DOI: 10.1109/TNN.2009.2015974.

[50] Janine Chen et al. "Data learning techniques and methodology for Fmax prediction". In: *IEEE International Test Conference (ITC)*. 2009, pp. 1–10. DOI: 10.1109/TEST.2009.5355620.

[51] Janine Chen et al. "Selecting the most relevant structural Fmax for system Fmax correlation". In: *28th VLSI Test Symposium (VTS)*. 2010, pp. 99–104. DOI: 10.1109/VTS.2010.5469604.

[52] Xiao Chen et al. "Heterogeneous Committee-Based Active Learning for Entity Resolution (HeALER)". In: Aug. 2019, pp. 69–85. ISBN: 978-3-030-28729-0. DOI: 10.1007/978-3-030-28730-6_5.

[53] Davide Chicco, Matthijs J. Warrens, and Giuseppe Jurman. *The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation*. en. July 2021. DOI: 10.7717/peerj-cs.623. URL: http://dx.doi.org/10.7717/peerj-cs.623.

[54] David Cohn, Les Atlas, and Richard Ladner. "Improving Generalization with Active Learning". In: *Mach. Learn.* 15.2 (May 1994), pp. 201–221. ISSN: 0885-6125. DOI: 10.1023/A:1022673506211. URL: %5Curl%7Bhttps://doi.org/10.1023/A:1022673506211%7D.

[55] B. D. Cory, R. Kapur, and B. Underwood. "Speed binning with path delay test in 150-nm technology". In: *IEEE Design Test of Computers* 20.5 (2003), pp. 41–45. DOI: 10.1109/MDT.2003.1232255.

[56] J.S. Cramer. "The Origins of Logistic Regression". en. In: *SSRN Electronic Journal* (2003). ISSN: 1556-5068. DOI: 10.2139/ssrn.360300. URL: %5Curl%7Bhttp://www.ssrn.com/abstract=360300%7D.

[57] Luca Demarchi et al. "Recursive Feature Elimination and Random Forest Classification of Natura 2000 Grasslands in Lowland River Valleys of Poland Based on Airborne Hyperspectral and LiDAR Data Fusion". In: *Remote Sensing* 12.11 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12111842. URL: %5Curl%7Bhttps://www.mdpi.com/2072-4292/12/11/1842%7D.

[58] Harris Drucker et al. "Support Vector Regression Machines". In: *Advances in Neural Information Processing Systems*. 1996. URL: https://proceedings.neurips.cc/paper_files/paper/1996/file/d38901788c533e8286cb6400b40b386d-Paper.pdf.

[59] EIT. *Types and Applications of Microcontrollers*. 2024. URL: https://www.eit.edu.au/resources/types-and-applications-of-microcontrollers/.

[60] Sophie M. Fosson. "A Biconvex Analysis for Lasso ℓ_1 Reweighting". In: *IEEE Signal Process. Lett.* 25.12 (2018), pp. 1795–1799.

[61] Sophie M. Fosson, Vito Cerone, and Diego Regruto. "Sparse linear regression from perturbed data". In: *Automatica* 122 (2020), p. 109284. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2020.109284. URL: %5Curl%7Bhttps://www.sciencedirect.com/science/article/pii/S0005109820304830%7D.

[62] Giulia Fracastoro et al. "Playing the Lottery With Concave Regularizers for Sparse Trainable Neural Networks". In: *IEEE Trans. Neural Netw. Learn. Syst.* (2024), pp. 1–11.

[63] Jerome H Friedman. "Multivariate adaptive regression splines". In: *The annals of statistics* 19.1 (1991), pp. 1–67.

[64] Gilles Gasso, Alain Rakotomamonjy, and StÉphane Canu. "Recovering Sparse Signals With a Certain Family of Nonconvex Penalties and DC Programming". In: *IEEE Transactions on Signal Processing* 57.12 (2009), pp. 4686–4698. DOI: 10.1109/TSP.2009.2026004.

[65] Stuart Geman, Elie Bienenstock, and René Doursat. "Neural Networks and the Bias/Variance Dilemma". In: *Neural Computation* 4.1 (1992), pp. 1–58. DOI: 10.1162/neco.1992.4.1.1.

[66] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[67] Isabelle Guyon and André Elisseeff. "An Introduction to Variable and Feature Selection". In: *The Journal of Machine Learning Research* 3 (Mar. 2003), pp. 1157–1182. ISSN: 1532-4435.

[68] Isabelle Guyon et al. "Gene Selection for Cancer Classification Using Support Vector Machines". In: *Machine Learning* 46 (Jan. 2002), pp. 389–422. DOI: 10.1023/A:1012487302797.

[69] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "The elements of statistical learning: data mining, inference, and prediction". In: *Springer Science and Business Media* (2009).

[70] M.A. Hearst et al. "Support vector machines". In: *IEEE Intelligent Systems and their Applications* 13.4 (1998), pp. 18–28. DOI: 10.1109/5254.708428.

[71] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (Apr. 1998), pp. 107–116. DOI: 10.1142/S0218488598000094.

[72] Victoria Hodge and Jim Austin. "A Survey of Outlier Detection Methodologies". In: *Artificial Intelligence Review* 22 (Oct. 2004), pp. 85–126. DOI: 10.1023/B:AIRE.0000045502.10941.a9.

[73] Arthur E. Hoerl and Robert W. Kennard. "Ridge Regression: Biased Estimation for Nonorthogonal Problems". In: *Technometrics* 12.1 (1970), pp. 55–67. DOI: 10.1080/00401706.1970.10488634. eprint: https://www.tandfonline.com/doi/pdf/10.1080/00401706.1970.10488634. URL: https://www.tandfonline.com/doi/abs/10.1080/00401706.1970.10488634.

[74] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. "Comparing images using the Hausdorff distance". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.9 (1993), pp. 850–863. DOI: 10.1109/34.232073.

[75] Daniel Jarrett et al. *HyperImpute: Generalized Iterative Imputation with Automatic Model Selection*. 2022. arXiv: 2206.07769 [stat.ML]. URL: https://arxiv.org/abs/2206.07769.

[76] Weikuan Jia et al. "Feature Dimensionality Reduction: a Review". In: *Complex & Intelligent Systems* 8.3 (June 2022), pp. 2663–2693. ISSN: 2198-6053. DOI: 10.1007/s40747-021-00637-x. URL: https://doi.org/10.1007/s40747-021-00637-x.

[77] Ian T. Jolliffe and Jorge Cadima. "Principal component analysis: a review and recent developments". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065 (Apr. 2016). DOI: 10.1098/rsta.2015.0202. URL: %5Curl%7Bhttp://dx.doi.org/10.1098/rsta.2015.0202%7D.

[78] M. I. Jordan and T. M. Mitchell. "Machine Learning: Trends, Perspectives, and Prospects". In: *Science* 349.6245 (July 2015), pp. 255–260. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aaa8415.

[79] Utkarsh Mahadeo Khaire and R. Dhanalakshmi. "Stability of feature selection algorithm: A review". In: *Journal of King Saud University - Computer and Information Sciences* 34.4 (2022), pp. 1060–1073. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2019.06.012. URL: https://www.sciencedirect.com/science/article/pii/S1319157819304379.

[80] Asifullah Khan et al. "A survey of the recent architectures of deep convolutional neural networks". In: *Artificial Intelligence Review* 53.8 (2020), pp. 5455–5516. DOI: 10.1007/s10462-020-09825-6. URL: %5Curl%7Bhttps://doi.org/10.1007%2Fs10462-020-09825-6%7D.

[81] Tobias Kilian et al. “A Scalable Design Flow for Performance Monitors Using Functional Path Ring Oscillators”. In: *2021 IEEE International Test Conference (ITC)*. 2021, pp. 299–303. DOI: 10.1109/ITC50571.2021.00041.

[82] Tobias Kilian et al. “An efficient High-Volume Production Performance Screening using On-Chip Ring Oscillators”. In: *2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2023, pp. 1–6. DOI: 10.1109/DFT59622.2023.10313542.

[83] JunYoung Kim et al. “Recognition of face orientation by divided hausdorff distance”. In: *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*. 2015, pp. 564–567. DOI: 10.1109/TSP.2015.7296326.

[84] Georgios Kostopoulos et al. “Semi-Supervised Regression: A Recent Review”. In: *Journal of Intelligent & Fuzzy Systems* 35 (2018). 2, pp. 1483–1500. ISSN: 1875-8967. DOI: 10.3233/JIFS-169689. URL: %5Curl%7Bhttps://doi.org/10.3233/JIFS-169689%7D.

[85] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105.

[86] L.Sunitha et al. “Automatic Outlier Identification in Data Mining Using IQR in Real-Time Data”. In: *International Journal of Advanced Research in Computer and Communication Engineering* 3.6 (2014).

[87] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning”. en. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539.

[88] M. Leshno et al. “Multilayer Feedforward Networks with a Non-Polynomial Activation Function Can Approximate Any Function”. In: *New York University Stern School of Business Research Paper Series* (1993).

[89] David D. Lewis and William A. Gale. “A Sequential Algorithm for Training Text Classifiers”. In: *CoRR* abs/cmp-lg/9407020 (1994). arXiv: cmp-lg/9407020. URL: %5Curl%7Bhttp://arxiv.org/abs/cmp-lg/9407020%7D.

[90] Bowen Li et al. *Minimax Concave Penalty Regularized Adaptive System Identification*. 2023. arXiv: 2211.03903 [eess.SP].

[91] Wei Li et al. “Hausdorff GAN: Improving GAN Generation Quality With Hausdorff Metric”. In: *IEEE Transactions on Cybernetics* (2021), pp. 1–13. DOI: 10.1109/TCYB.2021.3062396.

[92] Mingsheng Long et al. "Learning transferable features with deep adaptation networks". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. 2015, pp. 97–105.

[93] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in neural information processing systems* 30 (2017).

[94] Scott M Lundberg et al. "From local explanations to global understanding with explainable AI for trees". In: *Nature machine intelligence* 2.1 (2020), pp. 2522–5839.

[95] Michael Lustig, David L. Donoho, and John M. Pauly. "Sparse MRI: The Application of Compressed Sensing for Rapid MR Imaging". In: *Magnetic Resonance in Medicine* 58.6 (2007), pp. 1182–1195. DOI: 10.1002/mrm.21391.

[96] Qinzhe Lv et al. "Enhanced-Random-Feature-Subspace-Based Ensemble CNN for the Imbalanced Hyperspectral Image Classification". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 14 (2021), pp. 3988–3999. DOI: 10.1109/JSTARS.2021.3069013.

[97] Anqi Mao, Mehryar Mohri, and Yutao Zhong. *Cross-Entropy Loss Functions: Theoretical Analysis and Applications*. 2023. arXiv: 2304.07288 [cs.LG].

[98] Konstantinos Maragos, George Lentaris, and Dimitrios Soudris. "In-the-Field Mitigation of Process Variability for Improved FPGA Performance". In: *IEEE Transactions on Computers* 68.7 (2019), pp. 1049–1063. DOI: 10.1109/TC.2019.2898833.

[99] Prem Melville and Raymond J. Mooney. "Diverse Ensembles for Active Learning". In: *Proceedings of the Twenty-First International Conference on Machine Learning*. ICML '04. Banff, Alberta, Canada: Association for Computing Machinery, 2004, p. 74. ISBN: 1581138385. DOI: 10.1145/1015330.1015385. URL: https://doi.org/10.1145/1015330.1015385.

[100] Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. *Learning Functions: When Is Deep Better Than Shallow*. 2016. arXiv: 1603.00988 [cs.LG].

[101] Jose García Moreno-Torres, José A. Saez, and Francisco Herrera. "Study on the Impact of Partition-Induced Dataset Shift on k-Fold Cross-Validation". In: *IEEE Transactions on Neural Networks and Learning Systems* 23.8 (2012), pp. 1304–1312. DOI: 10.1109/TNNLS.2012.2199516.

[102] Szu-Pang Mu et al. "Statistical Framework and Built-In Self-Speed- Binning System for Speed Binning Using On-Chip Ring Oscillators". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.5 (2016), pp. 1675–1687. DOI: 10.1109/TVLSI.2015.2478921.

[103] Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[104] Giorgio Di Natale et al. *Cross-Layer Reliability of Computing Systems*. Jan. 2020.

[105] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[106] Ashley Prater-Bennette, Lixin Shen, and Erin E. Tripp. *The Proximity Operator of the Log-Sum Penalty*. 2021. arXiv: 2103.02681 [math.OC].

[107] Mihalis Psarakis et al. "Microprocessor Software-Based Self-Testing". In: *IEEE Design Test of Computers* 27.3 (2010), pp. 4–19. DOI: 10.1109/MDT.2010.5.

[108] J. Ross Quinlan. "Induction of Decision Trees". In: *Machine Learning* 1.1 (1986), pp. 81–106. DOI: 10.1007/BF00116251.

[109] Sebastian Raschka. *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*. 2020. arXiv: 1811.12808 [cs.LG].

[110] H. Raza, G. Prasad, and Y. Li. "Dataset Shift Detection in Non-stationary Environments Using EWMA Charts". In: *2013 IEEE International Conference on Systems, Man, and Cybernetics*. 2013, pp. 3151–3156. DOI: 10.1109/SMC.2013.537.

[111] Ryan M. Rifkin and Ross A. Lippert. "Notes on Regularized Least Squares". en. In: (May 2007). URL: https://dspace.mit.edu/handle/1721.1/37318.

[112] Jeremy Rogers and Steve Gunn. "Identifying Feature Relevance Using a Random Forest". en. In: *Subspace, Latent Structure and Feature Selection*. Ed. by Craig Saunders et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 173–184. ISBN: 9783540341383. DOI: 10.1007/11752790_12.

[113] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." en. In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519. URL: %5Curl%7Bhttp://doi.apa.org/getdoi.cfm?doi=10.1037/h0042519%7D.

[114] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. "Efficient Implementation of the K-SVD Algorithm using Batch Orthogonal Matching Pursuit". In: 2008. URL: %5Curl%7Bhttps://api.semanticscholar.org/CorpusID:3251546%7D.

[115] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv: 1609.04747 [cs.LG].

[116] T. Ruokonen and International Federation of Automatic Control. *Fault Detection, Supervision, and Safety for Technical Processes*. SAFEPROCESS'94 : IFAC Symposium, Helsinki University of Technology v. 2. International Federation of Automatic Control, 1994. ISBN: 9780080422220. URL: %5Curl%7Bhttps://books.google.it/books?id=anBRAAAAMAAJ%7D.

[117] M. Sadi et al. "SoC Speed Binning Using Machine Learning and On- Chip Slack Sensors". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 36.5 (2017), pp. 842–854. DOI: 10.1109/TCAD.2016.2602806.

[118] S.R. Safavian and D. Landgrebe. "A survey of decision tree classifier methodology". In: *IEEE Transactions on Systems, Man, and Cybernetics* 21.3 (1991), pp. 660–674. DOI: 10.1109/21.97458.

[119] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229. ISSN: 0018-8646. DOI: 10.1147/rd.33.0210.

[120] G. Sannena and B. P. Das. "Low Overhead Warning Flip-Flop Based on Charge Sharing for Timing Slack Monitoring". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.7 (2018), pp. 1223–1232. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2018.2810954.

[121] Patrick Schober, Christa Boer, and Lothar A. Schwarte. "Correlation Coefficients: Appropriate Use and Interpretation". In: *Anesthesia & Analgesia* 126 (2018), pp. 1763–1768.

[122] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. "Kernel principal component analysis". In: *Artificial Neural Networks — ICANN'97*. Ed. by Wulfram Gerstner et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 583–588. ISBN: 978-3-540-69620-9.

[123] Raphael Schumann and Ines Rehbein. "Active Learning via Membership Query Synthesis for Semi-Supervised Sentence Classification". In: *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*. Hong Kong, China: Association for Computational Linguistics, 2019, pp. 472–481. DOI: 10.18653/v1/K19-1044. URL: %5Curl%7Bhttps://www.aclweb.org/anthology/K19-1044%7D.

[124] ScienceDirect. "Microcontroller - an overview". In: *ScienceDirect Topics* (2024). URL: https://www.sciencedirect.com/topics/computer-science/microcontroller.

[125] A. G. Sefidmazgi, F. Ahmadi-Abkenari, and S. A. Mirroshandel. "Correlation analysis as a dependency measures for inferring of time-lagged gene regulatory network". In: *Eighth International Conference on Information and Knowledge Technology (IKT)*. 2016, pp. 6–11. DOI: 10.1109/IKT.2016.7777761.

[126] J. Shawe-Taylor and Shiliang Sun. "A review of optimization methodologies in support vector machines". In: *Neurocomputing* 74 (2011), pp. 3609–3618.

[127] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[128] "Spearman Rank Correlation Coefficient". en. In: *The Concise Encyclopedia of Statistics*. New York, NY: Springer, 2008, pp. 502–505. ISBN: 9780387328331. DOI: 10.1007/978-0-387-32833-1_379. URL: %5Curl%7Bhttps://doi.org/10.1007/978-0-387-32833-1_379%7D (visited on 11/28/2023).

[129] Eleftherios Spyromitros-Xioufis et al. "Multi-Label Classification Methods for Multi-Target Regression". In: *arXiv* (Nov. 2012).

[130] Eleftherios Spyromitros-Xioufis et al. "Multi-target regression via input space expansion: treating targets as inputs". In: *Machine Learning* 104.1 (Feb. 2016), pp. 55–98. ISSN: 1573-0565. DOI: 10.1007/s10994-016-5546-z. URL: %5Curl%7Bhttp://dx.doi.org/10.1007/s10994-016-5546-z%7D.

[131] Daniel J. Stekhoven and Peter Bühlmann. "MissForest—non-parametric missing value imputation for mixed-type data". In: *Bioinformatics* 28.1 (Oct. 2011), pp. 112–118. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btr597. eprint: https://academic.oup.com/bioinformatics/article-pdf/28/1/112/583703/btr597.pdf. URL: %5Curl%7Bhttps://doi.org/10.1093/bioinformatics/btr597%7D.

[132] Haralampos-G. Stratigopoulos. "Machine learning applications in IC testing". In: *2018 IEEE 23rd European Test Symposium (ETS)*. 2018, pp. 1–10. DOI: 10.1109/ETS.2018.8400701.

[133] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[134] Robert Tibshirani. "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1 (1996), pp. 267–288. ISSN: 00359246. URL: %5Curl%7Bhttp://www.jstor.org/stable/2346178%7D (visited on 11/27/2023).

[135] Simon Tong and Daphne Koller. "Support Vector Machine Active Learning With Applications To Text Classification". In: *JMLR* 2 (Dec. 2001), pp. 45–66. DOI: 10.1162/153244302760185243.

[136] Joel A. Tropp and Anna C. Gilbert. "Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit". In: *IEEE Transactions on Information Theory* 53.12 (2007), pp. 4655–4666. DOI: 10.1109/TIT.2007.909108.

[137] D. Tuia, E. Pasolli, and W. J. Emery. "Dataset shift adaptation with active queries". In: *2011 Joint Urban Remote Sensing Event*. 2011, pp. 121–124. DOI: 10.1109/JURSE.2011.5764734.

[138] D. Tuia et al. "A Survey of Active Learning Algorithms for Supervised Remote Sensing Image Classification". In: *IEEE JSTSP, Journal of Selected Topics in Signal Processing* 5.3 (2011), pp. 606–617. DOI: 10.1109/JSTSP.2011.2139193.

[139] Alan M. Turing. "Computing Machinery and Intelligence". en. In: *Mind* LIX.236 (1950), pp. 433–460. ISSN: 1460-2113, 0026-4423. DOI: 10.1093/mind/LIX.236.433. URL: %5Curl%7Bhttps://academic.oup.com/mind/article/LIX/236/433/986238%7D.

[140] V. Vapnik. "Principles of Risk Minimization for Learning Theory". In: *Advances in Neural Information Processing Systems*. Ed. by J. Moody, S. Hanson, and R. P. Lippmann. Vol. 4. Morgan-Kaufmann, 1992. URL: https://proceedings.neurips.cc/paper/1991/file/ff4d5fbbafdf976cfdc032e3bde78de5-Paper.pdf.

[141] P.N. Variyam, S. Cherubal, and A. Chatterjee. "Prediction of analog performance parameters using fast transient testing". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 21.3 (2002), pp. 349–361. DOI: 10.1109/43.986428.

[142] K. von Arnim et al. "An Effective Switching Current Methodology to Predict the Performance of Complex Digital Circuits". In: *IEEE International Electron Devices Meeting (IEDM)*. 2007, pp. 483–486. DOI: 10.1109/IEDM.2007.4418979.

[143] Kevin Vu et al. *Understanding Kernel Ridge Regression: Common behaviors from simple functions to density functionals*. 2015. DOI: 10.48550/ARXIV.1501.03854. URL: https://arxiv.org/abs/1501.03854.

[144] Li-C. Wang. "An Autonomous System View To Apply Machine Learning". In: *IEEE International Test Conference (ITC)*. 2018, pp. 1–10. DOI: 10.1109/TEST.2018.8624844.

[145] Li-C. Wang. "Experience of Data Analytics in EDA and Test—Principles, Promises, and Challenges". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 36.6 (2017), pp. 885–898. DOI: 10.1109/TCAD.2016.2621883.

[146] Quan Wang. *Kernel Principal Component Analysis and its Applications in Face Recognition and Active Shape Models*. 2012. DOI: 10.48550/ARXIV.1207.3538. URL: https://arxiv.org/abs/1207.3538.

[147] S. Wang et al. "Pool-based active learning based on incremental decision tree". In: *2010 ICMLC, International Conference on Machine Learning and Cybernetics*. Vol. 1. 2010, pp. 274–278. DOI: 10.1109/ICMLC.2010.5581052.

[148] Joe H. Ward. "Hierarchical Grouping to Optimize an Objective Function". In: *Journal of the American Statistical Association* 58.301 (1963), pp. 236–244. DOI: 10.1080/01621459.1963.10500845. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1963.10500845. URL: %5Curl%7B%20https://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500845%20%7D.

[149] Michael Weiss and Paolo Tonella. "Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study)". In: *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA '22. ACM, July 2022. DOI: 10.1145/3533767.3534375. URL: http://dx.doi.org/10.1145/3533767.3534375.

[150] R.H. Williams and C.F. Hawkins. "The Effect of Guardbands on Errors in Production Testing". In: *Third European Test Conference (ETC)*. 1993, pp. 2–7. DOI: 10.1109/ETC.1993.246539.

[151] Dongrui Wu, Chin-Teng Lin, and Jian Huang. *Active Learning for Regression Using Greedy Sampling*. 2018. arXiv: 1808.04245 [cs.LG].

[152] Lifeng Wu and Junliang Fan. *Comparison of neuron-based, kernel-based, tree-based and curve-based machine learning models for predicting daily reference evapotranspiration*. en. Ed. by Paweł Pławiak. May 2019. DOI: 10.1371/journal.pone.0217520. URL: http://dx.doi.org/10.1371/journal.pone.0217520.

[153] M. Wu and J. Ye. "A Small Sphere and Large Margin Approach for Novelty Detection Using Training Data with Outliers". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.11 (2009), pp. 2088–2092. DOI: 10.1109/TPAMI.2009.24.

[154] Huiliang Xie and Jian Huang. "SCAD-penalized regression in high-dimensional partially linear models". In: *The Annals of Statistics* 37.2 (2009), pp. 673–696. DOI: 10.1214/07-AOS580. URL: https://doi.org/10.1214/07-AOS580.

[155] Jie Yang and Jun Ma. "Compressive sensing-enhanced feature selection and its application in travel mode choice prediction". In: *Applied Soft Computing* 75 (2019), pp. 537–547. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2018.10.048. URL: %5Curl%7Bhttps://www.sciencedirect.com/science/article/pii/S1568494618306161%7D.

[156] Xiangli Yang et al. "A Survey on Deep Semi-Supervised Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 35.9 (2023), pp. 8934–8954. DOI: 10.1109/TKDE.2022.3220219.

[157] Jonathan Zarecki and Shaul Markovitch. "Textual Membership Queries". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence* (July 2020). DOI: 10.24963/ijcai.2020/369. URL: %5Curl%7Bhttp://dx.doi.org/10.24963/ijcai.2020/369%7D.

[158] J. Zeng et al. "On correlating structural tests with functional tests for speed binning of high performance design". In: *IEEE International Test Conference (ITC)*. 2004, pp. 31–37. DOI: 10.1109/TEST.2004.1386934.

[159] Xiaojin Zhu. *Semi-Supervised Learning Literature Survey*. Tech. rep. 1530. Computer Sciences, University of Wisconsin-Madison, 2005.

[160] Fuzhen Zhuang et al. "A Comprehensive Survey on Transfer Learning". In: *Computing Research Repository (CoRR)* abs/1911.02685 (2019). arXiv: 1911.02685. URL: %5Curl%7Bhttp://arxiv.org/abs/1911.02685%7D.

[161] Hui Zou and Trevor Hastie. "Regularization and Variable Selection via the Elastic Net". In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 67.2 (2005), pp. 301–320. ISSN: 13697412, 14679868. URL: http://www.jstor.org/stable/3647580 (visited on 12/18/2023).

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.