Politecnico di Torino

Politecnico di Torino

Master's Degree : Computer Engineering - Artificial Intelligence and Data Analytics

A.a. 2024/2025

Graduation Session October 2025

Optimizing Microcontroller Performance Screening through Data-Efficient Active Learning

An Experimental Study

Supervisors:

Riccardo Cantoro

Nicolò Bellarmino

Candidate:

Alessandro Vancini

Summary

Microcontrollers play a fundamental role in modern electronic systems, from consumer devices to critical domains, such as the automotive and aerospace industries. In order to ensure their reliability, microcontrollers undergo performance-based screening during production. Underperforming units are typically detected by measuring their maximum frequency under stressful operating conditions. However, such testing is both expensive and time-consuming. The screening phase typically provides only binary performance insights, but it could greatly benefit from the integration of real-valued information obtained with minimal effort through Machine Learning algorithms. This thesis explores the use of Machine Learning techniques to support and improve this screening phase, leveraging data from embedded Speed Monitors as predictors of device performance. A variety of regression models are evaluated and compared, with particular focus on how their accuracy evolves when trained with different amounts of data. To address the high cost of labeled samples, transfer learning strategies at database level are investigated, with the aim of constructing optimized training sets with minimal labeling effort. The final result of this experimentation is a set of learning curves that analyze the performance of different models, generated with incremental data availability and different sampling strategies, with the ultimate goal of improving predictive accuracy and reducing test costs in industrial settings.

Acknowledgements

I would like to sincerely thank Professor Riccardo Cantoro for giving me the opportunity to work on this thesis. I am especially grateful to my supervisor, Nicolò Bellarmino, for guiding me throughout these months, helping me overcome the many technical challenges I faced, and providing valuable personal advice. I also wish to thank Infineon Technologies AG for providing the data and information essential to this work, and for the insights shared during our meetings. Through the guidance of the Professor and my supervisor, and the collaboration with Infineon Technologies AG, this experience has been an important opportunity for personal growth, helping me develop skills and confidence that will undoubtedly be valuable in my future professional endeavors.

I would also like to thank my friends, who supported me, listened to my frustrations, offered advice, and, whether over a coffee, a game night, a simple walk, or keeping me company on Discord, helped me navigate truly stressful moments throughout my entire university journey.

A special thanks goes to my girlfriend, Sabrina, for being a pillar in my life over these years, giving me strength and peace of mind, and for even having the patience to follow my work and provide valuable suggestions, as well as offering moments of reflection and complete breaks from studying to recharge.

Finally, I owe the greatest gratitude to my parents and family, who made all of this possible from the very beginning. Despite moments of difficulty or misunderstanding, they always believed in me and maintained a positive attitude, even when I struggled to organize my life and studies. They worked tirelessly to give me the opportunity to succeed. Their patience, support, and sacrifices have allowed me to reach the end of this academic journey.

Finally, knowing this may sound inappropriate or even overly narcissistic, a huge thank you to myself for persevering through these years and continuing to move forward despite adversity.

Ringraziamenti (Versione Italiana)

Vorrei ringraziare sinceramente il professor Riccardo Cantoro per avermi dato l'opportunità di lavorare su questa tesi. Inoltre, sono particolarmente grato al mio relatore, Nicolò Bellarmino, per avermi guidato durante questi mesi, aiutandomi a risolvere le numerose sfide tecniche che ho dovuto affrontare e fornendomi preziosi consigli personali. Desidero inoltre ringraziare Infineon Technologies AG per aver fornito i dati e le informazioni fondamentali per questo lavoro, e per gli spunti condivisi durante i nostri incontri. Grazie alla guida del professor Cantoro e del mio relatore, e alla collaborazione con Infineon Technologies AG, questa esperienza è stata un'importante occasione di crescita personale, che mi ha permesso di sviluppare competenze e fiducia in me stesso che saranno senz'altro preziose nel mio futuro percorso professionale.

Desidero inoltre ringraziare i miei amici, che mi hanno supportato, ascoltato nei momenti di sfogo, dato consigli e, tra un caffè, una serata passata giocando, una semplice passeggiata o facendomi compagnia su Discord, mi hanno aiutato ad affrontare momenti davvero stressanti durante tutto il mio percorso universitario.

Un ringraziamento speciale va alla mia ragazza, Sabrina, per essere stata un pilastro nella mia vita in questi anni, donandomi forza e serenità, e per aver avuto la pazienza persino di seguire il mio lavoro e offrire suggerimenti preziosi, oltre a momenti di riflessione e di completo stacco dallo studio per ricaricare le batterie.

Infine, sono profondamente grato ai miei genitori e alla mia famiglia, che hanno reso tutto questo possibile fin dall'inizio. Anche se non sono mancati momenti di difficoltà o di incomprensione, hanno sempre continuato a credere in me e mantenere un atteggiamento positivo, anche quando faticavo a organizzare la mia vita e i miei studi. Hanno lavorato instancabilmente per darmi la possibilità di avere successo. La loro pazienza, il loro sostegno e i loro sacrifici mi hanno permesso di giungere al termine di questo percorso accademico.

Infine, consapevole di sembrare inopportuno o persino di sembrare eccessivamente narcisistico, un enorme grazie a me stesso per aver tenuto duro in questi anni e continuato ad andare avanti nonostante le avversità.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning is a rapidly developing field of research, driven by the strong attention it has received from the scientific community in recent years, and rightly so, as its impact and applications continue to expand. As a branch of the broader field of Artificial Intelligence, it is widely applied to support decision-making processes across numerous disciplines. Its effectiveness lies in the ability of models to automatically learn patterns from data and use them to generate predictions that can assist human experts in their tasks.

1.1 Problem Description

In this work, machine learning models with different configurations are employed to predict the performance of microcontrollers manufactured by Infineon Technologies A.G (specifically designed for the automotive field). These models are intended to be integrated into the production test flow to help achieve the security and performance standard. The performance of such microcontrollers is measured by finding their maximum operating frequency under worst-case voltage and temperature conditions.

Manufacturers of automotive microcontrollers, such as Infineon, must guarantee the reliability of the devices they produce. This means making sure that each unit can operate at the required minimum operating frequency even under disadvantageous environmental conditions. The automotive sector is particularly strict with quality standards, tolerating a maximum of 0.1 defective parts per million. Given that such constraints are fundamental, manufacturers have naturally adopted different screening processes to separate robust devices from those that do not meet performance requirements, ensuring that these sub-optimal units never reach the customer. However, carrying out these screening procedures is costly and time-consuming, as they require extensive testing to obtain a sufficient quantity of measurements under extreme conditions. In this work, the problem is addressed

through the exploration of machine learning models that operate on a reduced set of measurements, available from early-stage testing, to predict the maximum operating frequency.

The objective is to predict the worst-case maximum operating frequency with sufficient accuracy, while minimizing the need for exhaustive performance characterization. Ultimately, the models can be integrated into the production flow to improve efficiency and reduce costs, while maintaining the required reliability standards.

1.2 Goal

The goal of this work is to develop machine learning models to predict the *worst-case maximum operating frequency* of automotive microcontrollers manufactured by Infineon Technologies AG, where the prediction relies on measurements, collected in the early stages of production, by Speed Monitors (ring oscillators integrated in devices that capture physical quantities related to circuit performance). The models aim to establish a mathematical relationship between these measurements and the operating frequencies, enabling the generation of predictions of the frequency under critical voltage and temperature conditions. This approach shortens testing time and reduces resource usage while maintaining reliability, improving the performance screening process by providing detailed, real-valued estimates of the maximum operating frequency with minimal additional effort.

This study addresses several challenges:

- **Limited data availability**: Only a small number of devices and their associated measurements can be used for training, requiring careful selection of samples.
- **Production variability**: Data comes from non-stationary environments and inter-wafer variations, which can reduce prediction accuracy.
- **Multi-generation devices**: Older microcontroller generations are included to improve predictions on the latest generation of devices (A) leveraging transfer learning principles.
- **Lightweight model requirements**: Pipelines must be efficient to allow potential deployment directly on devices or in hardware-constrained environments, motivating Ridge as the most efficient option.

1.3 Related Work

The experimental foundation of this work derives from previous research (Computational Intelligence Techniques for Device Testing, Nicolò Bellarmino, 2021 [1]), which itself builds on earlier work on performance prediction from speed monitor measurements [2] and the application of active learning for training set optimization [3]. The fusion of these ideas led to the development of pipelines for model training and produced detailed analyses of PCA, dataset distributions, and multi-task learning. Furthermore, it highlighted the potential of lightweight models such as Ridge and Polynomial Ridge regressors. Expanding on these findings, the present study empirically evaluates predictive models, scaling strategies, and sample efficiency across various training strategies, data combinations, and preprocessing approaches.

1.4 Organization

The work is organized as follows: Chapter 2 provides background on automotive microcontrollers and the industrial testing process, with particular emphasis on early-stage measurements from speed monitors. Chapter 3 introduces the machine learning concepts relevant to this work, including regression methods, feature scaling, and strategies for handling dataset shift. Chapter 4 summarizes prior research that serves as the baseline and motivation for the present study. Chapter 5 describes the datasets, preprocessing pipeline, and methodology adopted, including scaling, learning curve strategies, and active learning approaches. Chapter 6 presents the main experiments and results, covering baseline models, comparisons between learning strategies, scaling techniques, wafer type distinctions, and feature similarity analyses. Finally, Chapter 7 summarizes the findings, highlights their implications for production testing, and discusses future research directions.

Chapter 2

Background: Microcontrollers and Testing

2.1 Microcontrollers

A microcontroller unit (MCU) can be thought of as a compact computer on a single chip, designed to handle dedicated tasks within embedded systems without the need for a full operating system. These integrated circuits incorporate a central processing unit (CPU), memory (both volatile RAM for temporary data and non-volatile flash or EEPROM), and a set of peripherals, such as timers, counters, analog-to-digital (ADC) and digital-to-analog (DAC) converters, and communication interfaces like UART, SPI, or I2C.

Unlike standard microprocessors, microcontrollers integrate processing, memory, and I/O functionality in a single chip, enabling efficient real-time control of devices. They are particularly suited for applications that require monitoring and signal processing, including controlling motors, reading sensors, and managing communication with other modules.

Automotive microcontrollers, in particular, are optimized for reliability, real-time processing, and operation under wide temperature and voltage ranges. They often include built-in diagnostic and safety features, along with specialized hardware modules like timers and on-chip monitoring structures like ring oscillators or Speed Monitors. These monitoring structures provide low-level measurements correlated with device performance, which can be exploited to predict operational characteristics and support production testing, removing the need for a full functional evaluation of every device.

Microcontrollers are widely applied in battery-powered consumer devices like smartphones and wearables, but also in professional environments such as automotive systems, industrial automation, robotics, and IoT solutions. They can be

programmed with languages such as C, C++, or Python, making them accessible to both developers working on personal or home projects, as well as engineers working on professional and industrial applications.

2.2 Reliability Assessment

To guarantee that devices perform as intended under critical operating conditions, manufacturers conduct systematic tests, including performance screening, which evaluates the maximum operating frequency under critical voltage, temperature, and task execution conditions. Units that do not meet performance criteria are discarded, ensuring that only compliant devices reach the customer. In the automotive sector, where quality standards are extremely high, and even a minimal fraction of defective units is unacceptable, extensive screening is required. This process relies on speed monitors, on-chip ring oscillators that capture physical quantities related to circuit performance, to efficiently identify underperforming devices. Traditionally, performance screening is carried out by executing dedicated test programs, which in this work are used to determine the maximum operating frequency. The use of speed monitors as predictive features, instead, represents a non-standard yet innovative approach forming the basis of a new framework being developed in collaboration with Infineon. This methodology allows the implementation of the so-called Alternate Test, which allows estimating key performance parameters, like the maximum frequency, from alternative but correlated measurements, such as the oscillation frequencies of the speed monitors.

2.3 Speed Monitors

As already mentioned, Speed Monitors (SMONs) are ring oscillators integrated in the microcontroller itself. More specifically, a ring oscillator consists of a cascaded chain of an odd number of inverters (logical NOT gates) arranged in a ring, meaning that the output of the last inverter in the chain is fed back into the first one. Since each inverter adds a small propagation delay to the signal, the signal continuously toggles, producing a stable square-wave oscillation. The frequency of this oscillation is highly sensitive to variation in the manufacturing process, voltage and temperature which directly affect the delay introduced by each inverter. Thanks to this sensitivity, Speed Monitors are capable of capturing critical information about the performance of the device at an early production stage. By placing multiple ring oscillators across a wafer, manufacturers can assess local variations in fabrication and detect underperforming units before proceeding further in the production pipeline. In the automotive context, ring oscillators provide a quick and low-cost method to perform performance screening, making

them one of the most versatile tools for chip evaluation. In this study, the evaluation of device performance relies exclusively on measurements obtained from Speed Monitors. The core assumption is that a direct relationship exists between the oscillation frequencies of these monitors and the maximum operating frequency of the microcontroller. Each device includes approximately 130 Speed Monitors, organized into identical modules distributed across different regions of the die to capture within-die variations. In earlier product generations, the number of Speed Monitors was limited to 27, categorized into five structural groups (INV, NAND, NOR, and VM). The increased number and distributed placement of SMONs in newer devices provide a more comprehensive characterization of the device, enabling more accurate performance modeling. Further information on SMON placement strategies and feature grouping for MCU performance screening can be found in [4, 5]. The information collected from the Speed Monitors serve as the primary input features for predicting the operational performance of each device, providing an indirect yet reliable estimation of performance through machine learning models.

2.4 Wafer Classification

In the context of wafer-based microcontroller production, it is important to distinguish between standard production wafers and split-lot wafers, as each serves a specific role. Production wafers represent the fully processed units, intended for distribution to the customer and mass fabrication. In contrast, split-lot wafers are a subset separated from production lots for engineering purposes: they are used to explore parameter spaces, test corner cases and evaluate design choices before implementation in full production. Therefore, these wafers are not intended to be reach the customer, but instead are used to study performance variations and possible optimizations. In this work, by including both production and split-lot wafers in the training set, it was possible to assess how the proportion of each type affects model accuracy on production, split-lot, and mixed test sets.

2.5 Testing

Testing of integrated circuits (ICs) is essential to ensure that devices meet the specifications declared in their datasheets, especially when it comes to maximum operating frequency and functional performance. Although testing is both time and cost intensive, it remains indispensable for delivering reliable devices. Therefore, an optimal trade-off between test coverage and production costs must be found.

IC testing occurs at multiple stages of the product's life-cycle. During the design phase, *validation tests* verify that the design can theoretically meet the specifications. Then, *characterization tests* are performed on the first prototypes

to identify potential failures, triggering design revisions if necessary. After these preliminary stages, large-scale production begins, with *on-wafer testing* performed before the chips are diced and packaged. Finally, *post-packaging tests* verify the correctness of the assembly process, and in some cases, in-field tests may be performed to validate the device under real operating conditions.

Beyond this life-cycle perspective, different types of tests target specific aspects of IC quality and reliability. *Wafer testing* is carried out in the early stages of production to identify defective chips before dicing. *Reliability testing* evaluates long-term device behavior by subjecting ICs to stress conditions to uncover potential degradation mechanisms. *Parametric testing* measures electrical characteristics such as voltage, current, and resistance, ensuring they remain within the specification. Finally, *memory testing* focuses on verifying correct operation of memory blocks and data storage functionality.

Test strategies can be broadly classified into two categories:

- **Functional tests**: These verify whether the IC performs according to its specifications by simulating real-world usage. This typically involve providing input to the IC, and then observing its output to confirm it behaves as expected, without relying on any knowledge of the internal structure. However, for modern digital microcontrollers, exhaustive functional verification has become infeasible due to the increasing complexity of their functionality.
- **Structural tests**: These aim to detect faults in the circuit itself rather than verifying every functional aspect. Structural testing generally involves checking individual components and their physical interconnections to ensure they are correctly placed and assembled according to the circuit's design. These tests may require additional external equipment and can be costly, particularly for devices with a large number of pins.

The key difference is that functional testing focuses on *what* the circuit does (its behavior and applications), whereas structural testing examines *how* it is built and connected.

To reduce costs and facilitate testing, features can be embedded into the chip itself. The design phase of the chip then also has to take into account the presence of required structures used for testing; this approach is also known as Design for Testability or Design for Testing (DfT). An alternative approach for microcontrollers is Software Based Self Test (SBST), which relies on executing tests through specific test programs that run inside the assembled system itself. These tests, executed by the processor, will exercise parts of the circuit and collect responses for evaluation making use of existing special instructions and performance monitoring mechanisms of the processor. The SBST approach is non-intrusive, does not require additional hardware, and runs at the processor's actual speed, allowing to detect defects that might be missed at lower speeds. Moreover, it provides a flexible and low-cost solution for testing embedded processors, especially those with limited circuitry dedicated to testing. However, despite the fact that SBST allows for a broader range of checks at lower cost and is easily applicable in the field, it is typically used to complement rather than replace traditional test strategies.

2.6 Application-Oriented SBST Programs

Within the SBST framework, the performance of each microcontroller is assessed by executing a set of application-oriented programs directly on the packaged die. These programs, referred to as functional patterns, are designed to emulate realistic customer workloads. For example, some reproduce complete applications, while others specifically target critical sub-circuits of the chip. In this way, all major functional aspects are exercised under representative operating conditions.

For each pattern, the operating frequency is gradually increased until a failure is observed. This procedure is repeated five times, and the median frequency is recorded to mitigate measurement noise. The evaluation relies on ten distinct patterns, ensuring a broad coverage of possible use cases. The lowest maximum frequency among them, referred to as the *critical pattern frequency*, is used as the decisive metric for device classification. If this value falls below a predefined performance threshold, the device is labeled as defective and excluded from shipment.

This methodology balances realism and precision by combining diverse functional scenarios with systematic frequency stepping. This allows manufacturers to obtain a reliable characterization of device performance that reflects both customer-oriented usage and critical hardware limitations.

Chapter 3

Background: Machine Learning

Machine Learning (ML) is a branch of artificial intelligence, whose core idea is to learn patterns from data rather than relying on explicit programming and modeling. Instead of relying on fixed rules or pre-defined mappings between input and output, ML systems improve their performance as they are exposed to more data. This makes them particularly suitable for complex problems where explicit modeling is infeasible. Over the past decade, ML has gained prominence across virtually all scientific and engineering domains, driven both by the exponential growth of available data and by advances in computational power and learning algorithms. Applications span diverse fields, from natural language processing and analysis of medical imaging, to autonomous driving. In each field, ML has proven capable of extracting meaningful representations from high-dimensional data and mapping them to accurate predictions in real-world scenarios. Within the semiconductor industry, and in particular for microcontroller testing, these same capabilities open new opportunities for performance prediction, test optimization, and reliability assessment.

Moreover, at its core, ML relies on the concept of training a model on a dataset and evaluating its ability to generalize to unseen data. The typical ML workflow generally consists of data collection, preprocessing, feature engineering, model training, and finally performance evaluation. A key factor in achieving good results is the quality and representativeness of the training data, since a model can only learn the patterns present in the training data it is exposed to. In this context, the notions of *overfitting* and *underfitting* describe the balance between memorizing training examples and learning patterns that allow the model to generalize well. A model is said to overfit when it adapts too closely to the properties and noise of the training data, achieving excellent accuracy on this or other data drawn from a

similar distribution, but poor performance on unseen data. Conversely, it is said to underfit when the model fails to capture the underlying relationships, leading to poor overall accuracy (both on training set and on unseen data). To address these challenges and improve robustness, ML engineers often rely on techniques such as feature scaling, dimensionality reduction, and regularization. These methods improve the ability of models to generalize and mitigate the issues arising from noise or high-dimensionality in the data.

In the semiconductor industry, and specifically in the testing of microcontrollers, ML has emerged as a valuable tool to face challenges posed by increasing circuit complexity and stringent performance requirements. Traditional test strategies, while effective, unfortunately suffer from high costs and limited flexibility and scalability when dealing with modern and more complex devices, which can have millions of transistors and a huge variety of applications. Therefore, given these limitations of traditional testing approaches, ML is becoming progressively more adopted and explored in field of microcontrollers testing, offering the potential of learning correlations between test data and device performance to predict key metrics (e.g. maximum operating frequency). This can significantly reduce test time and cost, while maintaining or even improving reliability.

In recent years, various studies have highlighted the potential of machine learning in the field of integrated circuit testing and semiconductor manufacturing ([6],[7],[8],[9]). These works discuss how ML can complement and, in some cases even outperform, traditional testing and analysis methods, especially when applied to industrial datasets and real-world case studies. Applications cover a broad range of tasks, including performance prediction, defect detection, process optimization, and reliability assessment. The relevance of ML grows as semiconductor technology evolves towards increasingly complex System on Chip (SoC) designs. Collectively, these studies illustrate that ML provides a powerful tool to extract meaningful patterns from the large volumes of structured data that are routinely generated, but often not fully exploited.

In the context of this thesis, machine learning provides the foundation for analyzing and predicting microcontroller performance based on functional test patterns. By leveraging regression models, dimensionality reduction techniques, and active learning strategies, the goal is to estimate the worst-case maximum operating frequency of a device (the lowest maximum frequency observed across all applied test patterns) using fewer measurements, thereby reducing the overall testing burden. Additional concepts such as domain adaptation and dataset-shift considerations are also relevant, as measurements from devices tested under varying conditions may follow significantly different distributions.

3.1 Core ML foundations

3.1.1 Supervised vs. Unsupervised Learning

Machine learning tasks are commonly divided into two broad categories:

- **Supervised learning:** The model is trained on data for which both inputs and corresponding outputs (labels) are known. The objective of the learning process is finding the mapping between input features and outputs. Typical supervised tasks include *classification*, where the goal is assigning an input to one of two or more categories; and *regression*, where the output is instead a continuous numerical value.
- **Unsupervised learning:** The model deals with unlabeled data, aiming to uncover hidden structures or groupings in the data, often through clustering or dimensionality reduction.

In the context of the present work, supervised learning is the central paradigm, since the available data consist of input features derived from functional test patterns together with corresponding labels indicating device performance.

3.1.2 Linear Regression

Linear regression is a fundamental supervised learning method used to model the relationship between a set of input variables $\mathbf{x} \in \mathbb{R}^D$ and a continuous output $y \in \mathbb{R}$. At its core, linear regression assumes that the output can be expressed as a linear combination of the inputs:

$$y = \mathbf{w}^\top \mathbf{x} + b + \epsilon, \tag{3.1}$$

where $\mathbf{w}$ are the weights, b is a bias term, and ϵ is an error term capturing the noise in the data. Formally, linear regression can be seen as a hypothesis space $h_\theta(\mathbf{x})$ parameterized by $\theta = (\mathbf{w}, b)$, and the optimal parameters are found by minimizing a cost function. The most common choice is the *mean squared error* (MSE) over the training dataset:

$$\text{MSE}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \Big(y_i - h_\theta(\mathbf{x}_i) \Big)^2, \tag{3.2}$$

which corresponds to the classical *least squares* solution. From a probabilistic perspective, assuming Gaussian noise on the outputs, minimizing the MSE is equivalent to maximizing the likelihood of the observed data under the model (*maximum likelihood estimation*). This dual interpretation allows linear regression to be understood in two complementary ways: as a deterministic optimization

problem, focusing on fitting parameters to minimize prediction error; and as a probabilistic model, focusing on explaining the distribution of outputs given inputs.

Extensions such as multiple outputs, regularization techniques, and Bayesian linear regression further expand the model's flexibility and robustness. These theoretical concepts, in addition to many others such as regularization techniques or the probabilistic vs deterministic interpretation, are described in detail in Bishop (2006, Ch. 3, pp. 137–143) [10].

3.1.3 Feature Scaling

Machine learning algorithms are often sensitive to the relative magnitude of input features. When features have different units or ranges, those with larger scales can dominate distance-based metrics (functions that measure how far apart two data points are in the feature space) or disproportionately affect optimization during training. Dealing with this problem is of utmost importance, since many ML algorithms rely on distance-based metrics (therefore on the geometry of the feature space) and a disparity in the scale of the features would completely compromise the learning process. Feature scaling refers to the process of transforming input variables so that they share comparable ranges or distributions mitigating this problem and improving the performance of the model, ensuring that all features contribute proportionally to the learning process. To be more precise, when input variables differ greatly in scale, optimization processes such as gradient descent may converge slowly or become highly biased towards those features with larger numerical ranges [10, Ch. 3].

A widely used approach is *standard scaling*, which transforms features to have zero mean and unit variance. Formally, given a feature x, its standardized form is obtained as

$$x' = \frac{x - \mu}{\sigma},$$

where μ and σ denote the mean and standard deviation of the feature across the training set. This procedure is implemented, for instance, in scikit-learn (an open-source Python library for machine learning and data analysis [11]) through the `StandardScaler` class [12]. Another common technique is *min–max normalization*, which linearly rescales data to a fixed interval such as [0,1].

In this work in particular, normalization is essential on a practical level since, without it, regression-based models could distort the relationships between features and device performance.

3.1.4 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a widely used dimensionality reduction technique that transforms a set of possibly correlated features into a set of linearly uncorrelated variables called *principal components* [13]. Given a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ with n samples and d features, PCA seeks a set of orthogonal vectors $\mathbf{w}_1, \ldots, \mathbf{w}_k$ (principal components) that maximize the variance of the projections of the data:

$$\mathbf{z}_i = \mathbf{X}\mathbf{w}_i, \quad i = 1, \ldots, k,$$

subject to orthonormality constraints

$$\mathbf{w}_i^\top \mathbf{w}_j = \delta_{ij},$$

where δ_{ij} is the Kronecker delta, equal to 1 if $i = j$ and 0 otherwise. Each vector $\mathbf{z}_i$ represents the projection of the original data onto the i-th principal component, capturing the maximum variance along that direction. The orthonormality constraints ensure that different components are uncorrelated and that each has unit length, avoiding redundancy in the captured information. The first component $\mathbf{w}_1$ captures the maximum variance in the data, the second captures the maximum remaining variance orthogonal to the first, and so on. By projecting the data onto the first k components, dimensionality is reduced while retaining most of the information.

In this work, PCA is applied to microcontroller test measurements to reduce the feature space, simplifying regression models. This not only allows more efficient training, but by filtering out redundant or noisy features, can additionally improve predictive performance, and also results in more lightweight models, as required by hardware-constrained environments.

3.1.5 Ridge Regression

Ridge regression, also known as Tikhonov regularization, is an extension of linear regression that introduces a penalty term on the magnitude of the model coefficients. Specifically, it is a regularization technique used to estimate regression coefficients when the independent variables are highly correlated or when the model is at risk of overfitting. By penalizing large coefficients, Ridge regression stabilizes the estimates and improves the model's generalization to unseen data. Formally, given a dataset with input features $\mathbf{X} \in \mathbb{R}^{n \times d}$ and output targets $\mathbf{y} \in \mathbb{R}^n$, the Ridge objective is

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left\{ \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_2^2 \right\},$$

where $\lambda \geq 0$ is the regularization parameter controlling the strength of the penalty. The vector $\boldsymbol{\beta}$ contains the weights assigned to each feature, and the

regularization term $\lambda\|\boldsymbol{\beta}\|_2^2$ discourages excessively large coefficients, helping to prevent overfitting and improving generalization. The first term is the least squares error, while the second term is responsible for penalizing large coefficients, reducing model complexity and mitigating overfitting, especially when features are highly correlated or the dataset is high-dimensional.

Ridge regression is widely used in practice because it provides more stable and generalizable models than ordinary least squares in these scenarios. It provides also robustness against correlations between features, improving again generalization to unseen data [10, Ch. 3, pp. 138–142].

3.1.6 Polynomial Features

Polynomial features are a technique used to extend linear models by introducing non-linear relationships between the original features. Given an input vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]^\top$, polynomial feature expansion generates new features by including all products of the original features up to a specified degree p. For instance, for degree $p = 2$ and two features x_1 and x_2, the expanded feature vector becomes

$$\phi(\mathbf{x}) = [1, x_1, x_2, x_1^2, x_1x_2, x_2^2]^\top.$$

Mathematically, polynomial regression can still be formulated as a linear regression problem in the transformed feature space:

$$y \approx \mathbf{w}^\top \phi(\mathbf{x}) + \epsilon,$$

where $\mathbf{w}$ are the weights in the polynomial feature space and ϵ represents the noise term. Given an original feature vector of dimension d and a polynomial degree p, the total number of features after expansion grows combinatorially as

$$N_{\text{features}} = \binom{d+p}{p} = \frac{(d+p)!}{d!\,p!},$$

The main advantage of using polynomial features is that they allow linear models to capture non-linear dependencies in the data while still benefiting from the convex optimization (minimization of a convex cost function, typically the mean squared error plus a regularization term) properties of linear models. However, increasing the polynomial degree may lead to higher-dimensional feature spaces, which can exacerbate overfitting and computational cost if not combined with regularization techniques.

In this work, polynomial features were explored, using the `PolynomialFeatures` function from the scikit-learn Python library [14] on microcontroller test measurements, to investigate whether capturing potential non-linear interactions between different functional test patterns and the worst-case maximum operating frequency could improve model performance.

3.2 Model evaluation and behavior

Properly evaluating a machine learning model is a crucial step to understand its predictive performance and generalization capability and identify opportunities for improving the model through adjustments in data, features, or model parameters. It represents a fundamental step for every machine learning engineer,in both research and practical applications.

In supervised regression tasks, common evaluation metrics include the mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and the coefficient of determination (R^2). Formally, given a dataset with n samples, inputs $\mathbf{x}_i$ and target outputs y_i, and model predictions $\hat{y}_i$, these metrics are defined as follows:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.3}$$

The mean squared error (MSE) measures the average squared difference between predicted and actual values.

$$\text{RMSE} = \sqrt{\text{MSE}} \tag{3.4}$$

The root mean squared error (RMSE) expresses the error in the same units as the target variable, making interpretation more intuitive.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.5}$$

The mean absolute error (MAE) gives the average magnitude of errors without squaring.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3.6}$$

The coefficient of determination (R^2) is a measure of quality of fit calculated as the proportion of variance in the dependent variable that can be predicted using the independent variables ($\bar{y}$ is the mean of the observed values). Moreover, (R^2) is a number between 0 and 1 which represents how well the model predicts an outcome.

Performance can be evaluated using different metrics, by using the appropriate method. For example, regression models use real numbers, while classification models use discrete or categorical labels. One of the most commonly used metrics is

accuracy, which measures the proportion of correct predictions, divided by the total number of predictions. For a binary classification problem, accuracy is computed as:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.7}$$

where TP and TN denote true positives and true negatives, while FP and FN denote false positives and false negatives, respectively. Accuracy is meaningful when the classes are well balanced. However, if the dataset is heavily imbalanced, accuracy can be misleading. For example, if 99% of the samples belong to the positive class and the model always predicts positive, accuracy would be 99%, despite the model failing to identify any negative samples.

To address this limitation, other metrics such as *precision*, *recall*, and the *F1-score* are introduced. Precision measures the proportion of predicted positive samples that are actually positive, and is calculated as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.8}$$

Precision is particularly important when it is critical to minimize false positive predictions.

Recall, on the other hand, measures the proportion of actual positive samples that are correctly predicted:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.9}$$

Recall is a key metric when the goal is to capture as many positive samples as possible, when missing true positives has a high cost. Possible applications include the classification of whether people have an illness or not, or in the context of microcontrollers, detecting as many defective devices as possible.

The F1-score combines precision and recall into a single metric, calculated as their harmonic mean:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.10}$$

The F1-score is useful when a balance between precision and recall is desired. In particular, it is effective in scenarios with imbalanced datasets, as it penalizes models that perform well on one metric but poorly on the other.

Beyond these fundamental metrics, the study of *model behavior* involves analyzing how performance evolves as a function of the amount of training data or model complexity. Such studies aim to understand how much "experience" the model requires to reach a desired level of accuracy. For this purpose, *learning curves* are a widely used tool: they plot performance metrics against the training set size, providing insights into whether additional data or feature engineering is required; and guide hyperparameter selection for optimal model tuning, to meet specific requirements.

3.3 Active Learning

Active learning is a machine learning paradigm in which the model is not trained on a fixed dataset, but actively selects the most informative samples from a pool of training data instead. The key idea is that not all data points are equally useful for training, therefore it is more optimal to focus on the most representative samples in order to improve model performance faster and with fewer labeled examples[15].

From a technical perspective, active learning strategies generally rely on a *query function* that measures informativeness, such as uncertainty sampling, margin sampling, or representativeness criteria. A commonly used notion of representativeness is based on the *Hausdorff distance*, which measures the distance between two sets, by capturing the largest deviation between any point in one set and the closest point in the other. In practice, this criterion guides the selection of new training samples that best represent *unexplored regions* of the feature space, ensuring both diversity and coverage of the training set. This incremental, data-efficient approach enables the study of how model performance evolves with training set size, under the assumption that the new data it is exposed to while building the learning curve is the most informative. Applying this to the context of microcontroller test measurements and performance prediction, such sampling strategies become particularly valuable. They make it possible to maximize predictive accuracy even when little data is available, as is generally the case in the early stages of the production process.

3.4 Handling domain/data differences

Models are often trained on data collected under specific conditions (the *source domain*) and later deployed in a *target domain*, which may be characterized by slightly different conditions. These discrepancies, commonly referred to as *dataset shift* or *domain shift*, can substantially degrade predictive performance if not properly addressed [16].

Technically speaking, dataset shift occurs when the joint distribution of inputs and outputs changes between training and deployment. Typical cases include *covariate shift*, where the distribution of features changes, and *label shift*, where the distribution of outputs differs. In the case of microcontroller test measurements, such variations arise across different product families, wafer lots, or measurement setups, rendering careful analysis and robust data handling essential.

A straightforward way to mitigate this problem is through scaling strategies, either applied globally or within each domain. Applying scaling facilitates the reduction of discrepancies in feature distributions, but despite being essential, it only represents the first simple step of the solution. Facing this challenge generally also requires the use of more advanced techniques that fall under the scope of *domain adaptation*, which explicitly seeks to align the source and target domains, to optimize the model performance as well as possible for its deployment. A widely studied example is the *Correlation Alignment (CORAL)* method, which matches the covariance structure of source and target features, thereby improving transferability [17].

By adopting such strategies, it becomes possible to ensure reliable model performance across heterogeneous data sources, a crucial requirement for performance prediction in production environments.

3.5 Improving models with limited data

When only a limited amount of labeled data is available, strengthening machine learning models through specific strategies becomes essential in order to try to achieve the required level of performance despite data scarcity. Among the main approaches for this specific situation, two are widely adopted: *data augmentation* and *transfer learning*.

Data augmentation refers to techniques that artificially expand the training set by introducing synthetic variations of the existing data, effectively creating additional, fictitious examples, designed to mimic plausible patterns in the original dataset. For example, when dealing with measurement data, this might involve adding controlled noise or transformations of the preexisting data, in order to preserve the underlying semantics while diversifying the feature space. Naturally, this is beneficial for the model, allowing it to achieve better generalization capabilities and robustness. This strategy is also applied when dealing with imbalanced classes in the available data, by constructing synthetic samples for the minority classes, as widely studied in [18].

Another powerful strategy is transfer learning, where knowledge gained from one task or domain (*source domain*) is reused to improve performance on a related but different task or *target domain* [19].

Formally, given a source domain $\mathcal{D}_S$ with task $\mathcal{T}_S$ and a target domain $\mathcal{D}_T$ with task $\mathcal{T}_T$, transfer learning aims to improve the predictive function $f_T(\cdot)$ for $\mathcal{T}_T$ in $\mathcal{D}_T$ by utilizing information from $\mathcal{D}_S$ and $\mathcal{T}_S$.

The main advantage of transfer learning is efficiency. By reusing previously acquired knowledge, models can adapt to new tasks with less labeled data and reduced computational effort. This also improves robustness, as prior exposure to related tasks or domains provides the model with richer representations that generalize better to the target domain. In the scope of this thesis, transfer learning can support the adaptation of predictive models across different microcontroller families, wafer lots, or testing conditions, enabling accurate estimation of frequencies (or other measured quantities from testing) even when only limited new data is available.

Chapter 4

Prior Work and Baseline Knowledge

The work presented in *Computational Intelligence Techniques for Device Testing* (Bellarmino, 2021) [1] provides the foundation for this thesis. The former study explored the use of machine learning to predict the worst-case maximum operating frequency of automotive microcontrollers, using speed monitor measurements. Preprocessing steps such as feature scaling and principal component analysis (PCA) were applied, followed by regression methods like Ridge and Polynomial Ridge regression, in order to capture correlations between test data and device performance. These pipelines demonstrated the potential of lightweight models in linking early-stage measurements to final performance, establishing the baseline knowledge on which the present work builds.

4.1 Problem Definition

The problem addressed in *Computational Intelligence Techniques for Device Testing* (Bellarmino, 2021) [1] is closely aligned with the context of the present thesis. Both works focus on predicting the worst-case maximum operating frequency of automotive microcontrollers from Speed Monitor measurements, a task motivated by the high cost and time demands of exhaustive testing. The key challenge is to exploit early-stage measurements to obtain reliable predictions of device performance under critical operating conditions, thereby reducing testing effort while meeting stringent quality requirements. The two studies are tightly connected by this shared context, with the present work representing a natural continuation of the previous study. While the earlier thesis established the foundations by exploring various models, addressing dataset-shift and multi-target learning, and developing preprocessing pipelines, the present work extends the analysis in a more

empirical direction, focusing on model behavior under multi-dataset training and active learning.

4.2 Key Contributions

The cited prior work conducted several analyses, focusing primarily on the following aspects:

- Development of regression models to predict the maximum operating frequency from low-level measurements obtained from Speed Monitors.
- Application of feature space reduction techniques (e.g. PCA) to mitigate redundancy and noise, ensuring lightweight and efficient models.
- Use of active learning strategies to optimize prediction performance in data-scarce scenarios.
- Implementation of domain adaptation and dataset-shift handling methods to account for process variations across wafer lots and product families.
- Systematic comparison of multiple models to establish robust baselines for predictive performance (Support Vector Classifier, Decision Tree Classifier, Random Forest Classifier, Multi-Layer Perceptron, Ridge Classifier, and a custom Neural Network).

4.3 Dataset and Modeling Overview

4.3.1 Data Collection and Preprocessing

The dataset used in the prior work is collected from split wafer lots in two phases. First, Speed Monitor values are measured while devices are still on the wafer and un-packaged. Then, functional patterns are applied to measure the maximum operating frequency for each device. Devices exhibiting physical or local defects are identified using the *Outlier Index*, calculated as:

$$\text{OutlierIndex} = \frac{n}{x} \tag{4.1}$$

where n is the number of tests in which the device is an outlier and x is the total number of tests. Only devices with OutlierIndex $= 0$ and with all ten label measurements available are retained for training. The resulting dataset consists of 2403 devices, each with 27 Speed Monitor features and 11 labels (10 functional patterns plus one artificial minimum frequency label) from 26 wafers. PCA is used to visualize wafer distributions in two dimensions.

4.3.2 Regressor Chain: Conceptual Overview

The *Regressor Chain* is a method used in multi-target regression that leverages the correlations among multiple output variables (labels) to improve prediction accuracy, originally proposed in the context of multi-label classification by [20] and later extended to regression.

The core idea is that, in multi-target problems where you have several dependent variables to predict simultaneously, instead of predicting each label independently, the prediction is performed sequentially. Therefore, each model in the chain predicts one label using not only the original input features but also the predictions of all previous labels in the chain.

The order of labels in the chain can influence performance, as earlier predictions impact subsequent ones. To mitigate sensitivity to label ordering, an Ensemble of Regressor Chains (ERC) can be used, averaging predictions from multiple chains with different label permutations.

Assuming multiple highly correlated labels, the main advantage of this approach is that it leverages the correlations between labels to improve predictive accuracy.

4.3.3 Machine Learning and Multi-Target Approaches

The screening methodology is designed to predict device performance using Speed Monitor measurements, $y = f(x)$, in order to determine whether a device meets a predefined performance threshold. Polynomial Ridge Regression serves as the baseline model, while additional models, including Random Forest and Support Vector Regression (SVR), are employed for error estimation and to support active learning. To predict the worst-case maximum operating frequency, two strategies are considered: a single-target approach, which selects the minimum frequency among the 10 patterns as a single label; and a multi-target approach, which leverages all pattern labels simultaneously to exploit correlations among them. Correlation analysis confirms that the labels are highly correlated, motivating the use of multi-target learning. Two multi-target strategies are applied: the Regressor Chain, which sequentially predicts labels using prior predictions; and Multi-Target Regressor Stacking, a two-stage approach. Any single-target model can serve as the base learner in these frameworks, allowing flexible combinations of linear and non-linear regressors while taking advantage of label correlations to improve prediction accuracy.

Preliminary experiments with the Regressor Chain confirmed that chaining single-target models can improve prediction performance over non-chained approaches. Using the scikit-learn implementation, a standard regression model (Polynomial Ridge or Random Forest) is placed in each step of the chain, with 5-fold cross-validation applied to avoid using true labels from prior steps. The approach

demonstrated improvements not only for the final label, representing the minimum operating frequency ($P_{\min}$), but also for intermediate labels.

An important factor is the order of regressors in the chain, since early predictions influence subsequent ones. To mitigate performance variability due to chain order, an Ensemble of Regressor Chains (ERC) was tested, averaging predictions from multiple chains with random permutations. This yielded modest additional improvements but increased computational cost, so ERC was not used extensively in later experiments.

A brief evaluation of a Two-Stage regression approach, using Polynomial Ridge as the weak learner, showed inferior performance compared to the Regressor Chain and was therefore not pursued further.

4.3.4 Active Learning and Dataset-Shift Considerations

Preliminary studies of active learning focused on selecting heterogeneous models for the Query-by-Committee (QBC) approach, including Polynomial Ridge, SVR, Random Forest, and Gaussian Process Regressor. For the experiments, Polynomial Ridge, Random Forest, and SVR were used due to their strong baseline performance. The objective is to exploit model diversity to guide sample selection efficiently in data-scarce scenarios.

A critical challenge arises from dataset shift, where the distribution of new production data differs from the training data. Since devices originate from 26 distinct wafers, standard random train-test splits may under-represent certain wafers. Additionally, process variations and time-dependent factors create a non-stationary environment, potentially reducing model reliability on unseen data.

To address this, a rigorous training/test split strategy was implemented using `LeavePGroupsOut` from scikit-learn library [21], leaving 1–5 wafers out at a time. This resulted in 397,150 unique train-test combinations. For each split, seven unsupervised distance-based metrics were computed to quantify the divergence between training and test sets. These features were then used to train a model, predicting the potential prediction error on unseen wafers.

Direct regression of the mean squared error (MSE) proved challenging, so errors were discretized into an *Error Zone* with three categories: Green (MSE < 50), Yellow (50–150), and Red (> 150). This categorization reflects increasing uncertainty in predictions, with Red samples requiring additional analysis. The resulting dataset consists of 397,150 points with 7 features and one categorical label, however, it was highly unbalanced (64% Green, 29% Yellow, 6% Red), which must be considered when evaluating model performance.

4.4 Experimental Results and Considerations

- Ridge and Polynomial Ridge Regression emerged as the most effective baseline due to their strong predictive power and reduced complexity. Hence, this forms the foundation for the present work, but with a multi-dataset approach.
- Multi-task learning leveraged correlations among target variables to improve critical pattern estimation, while Regressor Chain reduced nRMSE by nearly half a percentage point for most regression algorithms, although tree-based models saw limited gains.
- Non-Linear models (Random Forse, XGBoost) captured more complex relationship, at the expense of efficiency.
- Active learning proved effective in reducing the number of labeled samples needed to reach a target error, with LOF and Hausdorff Distance being the most reliable selection metrics.
- To mitigate dataset-shift, a novel *Error Zone* approach was introduced to estimate prediction reliability on new unlabeled wafers. Random Forest classifiers achieved up to 98% F1-Score, and custom neural networks matched this performance, but additionally offer the benefit of on-board mathematical representation.

Overall, the results from the prior work [1] demonstrated that machine learning can effectively reduce the cost and time of MCU performance screening, by reducing the reliance on costly exhaustive functional testing. In addition, it established a methodological foundation (scaling, PCA, Ridge, and Polynomial Ridge) as a baseline ML pipeline, and it showed how active learning and adaptation methods can extend these result to realistic production environments, where data vary in both quantity and characteristics over time.

Chapter 5

Methodology and Data

5.1 Datasets and Preprocessing

5.1.1 Overview of Data

The datasets employed in this work consist of measurements collected from on-chip Speed Monitors (integrated in the microcontrollers), during the device testing process. Alongside these measurements (features), each tested device is associated with a set of function labels, corresponding to its measured operating frequency under different test conditions (functional patterns). The number of functional patterns varies by dataset, ranging from 8 to 10.

The final label used for training and evaluation is defined as:

$$F_{\text{worst}} = \min_{p \in \mathcal{P}} F_p,$$

where $\mathcal{P}$ denotes the set of functional test patterns and F_p is the maximum frequency measured under pattern p. This value represents the worst-case maximum operating frequency, and it is the target variable in the employed machine learning models, which are designed to support performance screening. This is achieved by ensuring that each product reliably meets the minimum operating frequency, declared in the datasheet, across all possible practical scenarios. By focusing on the most conservative estimate (*worst-case scenario*), the prediction task aligns directly with the strict reliability requirements of the manufacturing process.

It is important to note that the datasets considered in this work consist of devices belonging to the same family. The similarity among the devices ensures that the measured features and functional labels are comparable across samples, making the development of machine learning models feasible. Had the datasets included devices with large architectural or functional differences, the relationships between features and performance could vary substantially, potentially preventing the creation of

accurate predictive models. This inherent similarity within each family provides a solid foundation for learning reliable mappings from Speed Monitor measurements to device performance.

The available data, however, is limited in size and heterogeneous in nature. For this reason, before being used for modeling, it undergoes a dedicating pruning process whose aim is to remove invalid or non-informative entries. This includes discarding samples with missing or inconsistent measurements, eliminating wafers represented by fewer than three samples, and filtering out features that are either deemed unrelated to the considered test configurations or contain non-valid information. The actual number of retained samples and features after data pruning differs across datasets and will be reported in the subsections below.

The datasets employed are four: dataset A, which corresponds to the microcontroller family used as the reference target for performance prediction; and datasets B, C and D, which correspond to older microcontroller families. Datasets B–D serve as complementary sources of information to dataset A, enabling multi-dataset training and transfer learning experiments. All the models developed in this work are evaluated in terms of their ability to predict the worst-case maximum frequency

$$F_{\text{worst}}$$

on samples from dataset A, making the latter the central benchmark for the study.

5.1.2 Data Cleaning and Pruning

Dataset A

The raw dataset initially consists of 427 samples and 144 features (Speed Monitor measurements, and other data) together with the functional frequency labels. As previously mentioned, the raw data contains non-valid measurements and underrepresented wafers. Therefore, data pruning was performed to prepare the dataset for the models, resulting in the final dataset containing 412 valid samples and 122 features.

Datasets B–D

Similar to Dataset A, each dataset originally comprises Speed Monitor measurements as features and functional frequency labels as targets.

Prior to modeling and just like dataset A, all datasets underwent a dedicated pruning process to ensure quality and consistency. Additionally, for consistency with Dataset A, only the worst-case maximum operating frequency (F_{worst}), defined as the minimum among all functional pattern labels, was retained as the target variable.

After pruning, the datasets were reduced as follows:

- **Dataset B**: reduced from $(1963, 537)$ to $(1642, 378)$ features and samples.
- **Dataset C**: reduced from $(2002, 819)$ to $(1244, 388)$ features and samples.
- **Dataset D**: reduced from $(1032, 731)$ to $(1030, 434)$ features and samples.

These preprocessing steps ensure that all datasets contain valid, informative samples suitable for training and evaluation. The retained data preserves the structure necessary to predict F_{worst}, supporting subsequent machine learning experiments and comparisons across microcontroller families.

Table 5.1 summarizes the final number of samples and features retained for each dataset after preprocessing.

Dataset	Raw Samples	Raw Features	Final Samples	Final Features
A	427	144	412	122
B	1963	537	1642	378
C	2002	819	1244	388
D	1032	731	1030	434

Table 5.1: Dataset sizes before and after data pruning.

As shown, the difference in sample sizes between the target dataset A and the others is significant, highlighting the need for careful model handling when making predictions on A, where data is more limited. This scenario also reflects in general real-world production conditions, in which new devices typically have fewer available measurements compared to older generations.

5.1.3 Feature Preprocessing

Feature Scaling

Feature scaling is a critical preprocessing step in this work, as the measured values from Speed Monitors can vary widely in magnitude. Scaling is particularly important for linear models such as Ridge Regression, which form the baseline in this work. In linear models, the optimization of the cost function is sensitive to the relative magnitude of features and if it is not standardized, the parameter estimation could be biased towards features with bigger magnitude, especially with larger numerical ranges. Standardization ensures that all features contribute equally to the model fitting.

Standardization (StandardScaler)

The baseline model in the majority of experiments, Ridge Regression and Polynomial Ridge Regression, relies on standard scaling of the input features. Standardization transforms each feature to have zero mean and unit variance, which is particularly suitable for linear models and regression-based approaches.

In the case of multi-dataset setting, this approach has been extended to a *per-product strategy*, where each product type (or microcontroller model) is scaled independently, ensuring that intra-product feature distributions are preserved. The transformation was performed using `StandardScaler` [22] which computes the mean and standard deviation from the training data, and applies the scaling consistently to the test set.

Alternative Scaling Strategies

In addition to standardization, other scaling techniques were briefly explored for further experiments to assess their impact on model performance:

- **Min-Max Scaling:** Scales each feature to a fixed range, typically [0,1], using `MinMaxScaler` [23].
- **Robust Scaling:** Scales features using statistics that are robust to outliers (median and interquartile range), implemented via `RobustScaler` [24].
- **Quantile Transformation:** Maps the feature distribution to a uniform or normal distribution, using `QuantileTransformer` [25].
- **Max-Absolute Scaling:** Scales features by their maximum absolute value, preserving sparsity in the dataset, via `MaxAbsScaler` [26].

These alternative scaling strategies were tested in experiments beyond the baseline, and specifically on the configuration with the best performance, to check whether substituting the standardization with a different scaling strategy could result in higher predictive performance.

5.1.4 Feature Space Reduction (PCA)

As in the baseline regression model pipeline introduced by [1], PCA is applied after standardization (scaling) of the features. This step is essential given the large number of features (5.1), which otherwise would lead to high computational costs.

The number of retained components was determined empirically, by finding a compromise between the preservation of predicting performance, and maintaining the model sufficiently lightweight for the hardware constraint of the environment.

Following the baseline work, the number of components is set to 32 for Ridge Regression and to 20 for Polynomial Ridge Regression. The implementation adopted is `PCA` from scikit-learn [27].

In multi-dataset setups, PCA was applied globally across all products. No per-product strategy was adopted.

5.1.5 Feature Expansion

Polynomial Features

To increase the representational power of linear models, the baseline methodology also considered polynomial feature expansion. In this work, polynomial features of degree 2 were considered.

Trade-off: Expressivity vs. Complexity

Introducing polynomial features increases the expressive capacity of linear models, enabling them to approximate non-linear dependencies that may exist between functional test measurements and device performance. However, this comes at the cost of a rapidly growing feature space, which increases both computational demands and the risk of overfitting. Moreover, such non-linear relationships may not always be strong enough to justify the added complexity. For this reason, polynomial ridge regression was explored and directly compared to the baseline Ridge regression without feature expansion.

To mitigate the dimensionality explosion, polynomial expansion was combined with PCA for feature space reduction. Following the results of the prior work [1], the feature space was reduced to 20 principal components, which was found to provide the best balance between expressivity and efficiency. This trade-off ensures that the resulting models remain lightweight and computationally feasible, while still potentially benefiting from the richer representational capacity of polynomial features.

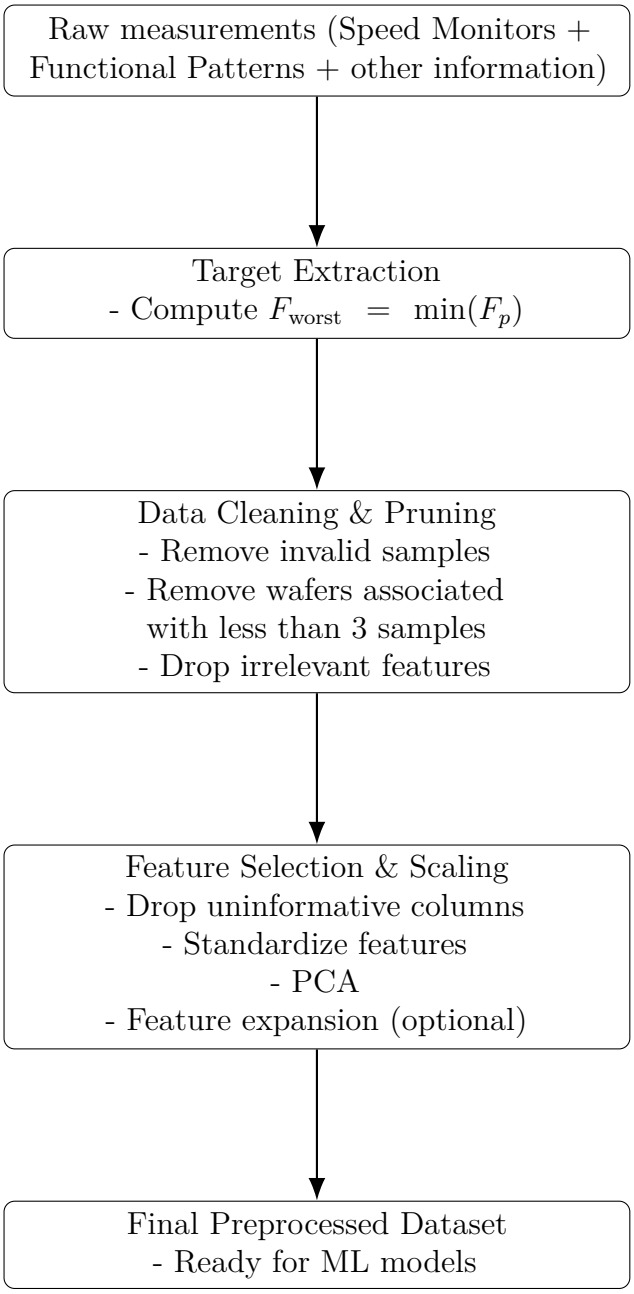

Figure 5.1: Preprocessing pipeline for microcontroller test data.

5.2 Train-Test Split Strategy

To evaluate model performance in a robust reproducible way, the dataset was divided into training and test sets. A test size of 20% of the available samples was selected, while the remaining 80% were used for training. The split was stratified according to the `LOT_Waf` identifier (a combined identifier linking each wafer to its production lot), ensuring that wafers with different lot associations were proportionally represented in both training and test sets. This stratification reduces the risk of having certain wafers entirely excluded from either of the subsets, which could bias the evaluation and compromise fairness during training.

Moreover, to account for variability due to random partitioning, each experiment was repeated over multiple splits using five different random seeds (`42, 16, 8, 9, 20`). The reported results are thus aggregated across these iterations, improving the statistical reliability of the performance comparison between models and setups.

The implementation of the split procedure relies on the `train_test_split` function from the `scikit-learn` library [28].

5.3 Learning Curve Strategies

To evaluate model performance as a function of the amount of training data, all experiments were conducted using an *incremental learning strategy*. Learning curves were generated by progressively increasing the number of training samples, providing insights into how predictive accuracy evolves as more data becomes available. This procedure reflects the practical scenario in production, where initially only a limited number of labeled samples for new devices are available. To emphasize the importance of dataset A in training, sample weights were applied in all incremental strategies, such that samples from dataset A received a weight of 1.0, while samples from older generations were assigned near-zero weights (0.0001). This was implemented to ensure that the model primarily learned from target-device data, while the data from older-generation devices was used as auxiliary information.

5.3.1 Incremental Strategies

Two main incremental strategies were investigated:

1. **Mixed Incremental Strategy:** All available samples, including older and newer generation devices, were shuffled and incrementally added to the training set. Standard preprocessing steps, such as feature scaling and the use of sample weights, were applied, but in an equal way across all device types. Experiments using this approach generally produced lower performance, as

the heterogeneity of the mixed training set supposedly reduces the model's ability to capture target-specific patterns. *For simplicity, this approach is referred to as "Strategy 1" in the following sections.*

2. **Old-Generation Baseline-First Incremental Strategy:** All samples from older generation devices were included in the training set from the beginning, serving as a baseline. Samples from the target microcontroller family (Dataset A) were then added incrementally. This strategy leverages knowledge from older devices while gradually incorporating target-specific information, consistently producing superior results compared to the mixed strategy. *For simplicity, this approach is referred to as "Strategy 2" in the following sections.*

5.3.2 Experimental Procedure for Learning Curves

Learning curves were generated to analyze model performance as a function of training set size, using the incremental strategies mentioned above. For each iteration, the following steps were performed:

1. **Train-Test Split:** The dataset was split into training and test sets, stratified by wafer labels. However, the test set contained only the target dataset to specifically evaluate the performance of the new-generation devices.

2. **Scaling:** The training set was expanded incrementally according to the chosen strategy. Target labels were standardized using `StandardScaler` fitted on the current training subset. In the experiments in which multiple device types were present, per-product scaling was applied, while global scaling was performed in the others. Sample weights, reflecting device type importance, were optionally applied.

3. **Model Training:** Models were trained using predefined pipelines, including Ridge and Polynomial Ridge regressors. Pipelines optionally included feature scaling and PCA transformations. Sample weights were incorporated in the training.

4. **Prediction and Evaluation:** Predictions were made on the test set restricted to the target dataset. Standardized predictions were inverse-transformed to the original target scale. Performance metrics were computed, including normalized root mean squared error (nRMSE), standard deviation of errors (Err-STD), as well as others (MSE, MAE, MAPE).

5. **Aggregation for Learning Curves:** Metrics were saved for each training fraction and random seed. For plotting, values were grouped by model type

and training set size, and averaged over random seeds to generate smoothed learning curves. The resulting plots included:

- nRMSE vs. training set size (logarithmic scale)
- Err-STD vs. training set size (logarithmic scale)

These plots allowed evaluation of model convergence and the impact of incrementally adding samples from the target dataset.

This protocol ensures that learning curves reflect the influence of incremental training, per-product scaling, sample weighting, and stratified evaluation, providing a reliable assessment of model performance on new-generation devices.

5.4 Active Learning Setup

Active learning (AL) experiments were performed in parallel with the incremental learning curve studies, following a phased approach to explore its effectiveness under different conditions.

1. **Strategy 1 Experiments with AL:** The first incremental strategy was initially tested on the target dataset (Dataset A) alone, applying active learning to select samples based on model uncertainty. Subsequently, Dataset A was combined with older generation datasets in various combinations, and the first incremental strategy was applied again, incorporating active learning in all cases.

2. **Strategy 2 Experiments without AL:** The second incremental strategy, which became the primary method after producing excellent results, was first executed without active learning. These experiments produced the best-performing learning curves and served as the baseline for further AL tests.

3. **Strategy 2 Experiments with AL:** Active learning was later applied to the best curves obtained from Strategy 2 experiments. The aim was to assess whether selective sampling from the target dataset could further enhance model performance, keeping the incremental setup.

4. **Additional Active Learning Experiments:** Further experiments were conducted using active learning guided by feature similarity analysis. These tests focused on a restricted set of the most similar features across datasets, applying Strategy 1 with active learning, and Strategy 2, first without active learning, and then applying active learning only to the best-performing configurations.

This structured approach ensures that the impact of active learning can be evaluated both in isolation and in combination with the different incremental strategies, providing a clear comparison of baseline performance and potential improvements.

For each iteration in the incremental training process, the training set was expanded fractionally, according to a predefined sequence of train sizes. The initial subset of samples was taken directly from the available training pool. Subsequent samples were selected using a *Hausdorff distance*-based approach, which measures the similarity between the current training set and the remaining candidates in feature space. The sample with the maximum Hausdorff distance to the current training set was chosen iteratively, ensuring that newly added samples contributed maximal diversity and information to the model.

5.5 Experimental Design

Given the methodology described in the previous sections, this section summarizes the experimental setup and outlines how the experiments were structured practically.

The experiments were organized to assess the impact of incremental strategies, active learning and feature selection on model performance, in addition to enabling comparisons across dataset combinations. Furthermore, sample weights, as defined in the section describing the incremental strategies (section 5.3.1), were incorporated during model training to prioritize dataset A samples.

The experiments can be categorized into the following groups:

1. **Full-Dataset Training Experiments:**
    - Evaluation of models trained on full datasets.
    - Performed first on dataset A alone, then with combinations of the latter with older datasets (B-D).
2. **Strategy 1 Experiments:**
    - Performed with active learning.
    - Applied to Dataset A alone, then in combination with older datasets (B-D).
3. **Strategy 2 Experiments:**
    - First performed without active learning to identify the best-performing incremental configurations. Followed by a second stage applying active learning to the optimal configurations to assess whether selective sampling could further improve model performance.

- Applied to combinations of datasets since *Strategy 2* cannot be applied to dataset A alone, as it relies on older-generation devices serving as a baseline training set.

4. **Dataset Similarity Analysis Experiments:**
   - Brief analysis of similarities among datasets in terms of feature and prediction target distributions (through Jensen-Shannon Divergence).
5. **Scaling Experiments:**
   - Testing different feature scaling strategies on the best-performing configuration, including MinMaxScaler, RobustScaler, and MaxAbsScaler.
   - Aiming to evaluate whether substituting the standard scaling with an alternative could further improve predictive performance.
6. **CORAL Experiments:**
   - Applying CORAL (Correlation Alignment) to the best-performing setup to align feature distributions across datasets in multi-dataset training.
   - Aiming to determine whether distribution alignment could provide additional performance gains when combining older generations and the target dataset.
7. **Wafer-distinction Based Training Experiments:**
   - Performing incremental training based on the distinction between production wafer samples and split-lot samples
   - Analyzing the performance in terms of accuracy (mainly nRMSE) on three different test sets : one consisting in mixed samples (both production and split-lot wafers samples), one consisting of production wafers samples only, and one made up of split-lot wafer samples only.

5.5.1 Evaluation Metrics and Methods

- Performance metrics included normalized root mean squared error (nRMSE), standard deviation of errors (Err-STD), and additional regression statistics (MSE, MAE, MAPE, R^2). Err-STD and nRMSE were the primary metrics used for plotting.
- Each experiment was repeated across five random seeds to account for variability in train-test splits. Similarly, metrics were aggregated (averaged) over random seeds for each training fraction, in order to generate smoothed learning curves and provide a more reliable measure of performance.

- Plots of nRMSE and err-STD versus training set size (number of samples from dataset A) were used to evaluate convergence and the effect of incrementally adding samples from the target dataset.

5.5.2 Comparisons and Analysis Objectives

The experiments were designed to meet the following goals of this thesis:

- Evaluate the relative performance of Strategy 1 versus Strategy 2.
- Assess the effect of active learning in the best configurations found with incremental strategies.
- Analyze briefly the similarity across older-generation datasets and target dataset A.
- Determine the impact of adding older-generation samples versus focusing solely on the target dataset (Dataset A).

Chapter 6

Experiments and Results

This chapter evaluates the performances of the previously described experiments, followed by a detailed report of the resulting data and outcomes of all of the conducted experiments.

The analysis begins with baseline experiments on the target dataset A, providing a reference point for model performance without incremental sampling and multi-dataset strategies. Next, experiments involving multiple datasets are presented, first through full-dataset training evaluations, subsequently comparing incremental learning strategies via learning curves. The best-performing setups are then revisited with active learning to assess the impact of selective sampling. Next, some complementary experiments are presented, covering scaling methods and wafer-type distinctions, which despite not being central to the thesis, still provide useful insights. Finally, the chapter concludes with a discussion of the results, highlighting the most effective strategies and most relevant findings.

The results are presented in full, displaying the baseline performance on the full training set, learning curves (nRMSE and Err-STD vs. training set size), and key observations.

6.1 Baseline Performance on Dataset A

6.1.1 Full-Dataset Training Evaluation

In this preliminary comparison, three regression algorithms were evaluated to establish a baseline performance using the target dataset A alone. Each model was trained following a specific pipeline configuration:

- **Ridge Regression:** StandardScaler → PCA (32 components) → Ridge regressor.

- **Polynomial Ridge Regression:** StandardScaler → PCA (20 components) → polynomial feature expansion (degree 2) → Ridge regressor.

- **TabPFN:** A modern transformer-based regression model included here for reference, to assess how a more complex approach compares to the lightweight models.

In each model, train-test splits were stratified by `LOT_Waf`, with random state fixed at 42 to ensure reproducibility. Moreover, all experiments were performed by training on the entirety of dataset A and evaluating on the corresponding test set. Table 6.1 reports normalized and relative metrics.

Algorithm	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	0.9478	1.12%	1.50%	1.12%	7.609
Poly Ridge	0.9451	1.20%	1.53%	1.21%	7.869
TabPFN	0.9435	1.22%	1.55%	1.22%	7.947

Table 6.1: Normalized/relative performance metrics (baseline models - trained on entirety of dataset A).

The scatter plots below (figs. 6.1 to 6.3) display the actual versus predicted target values for visual assessment of prediction quality by each model.

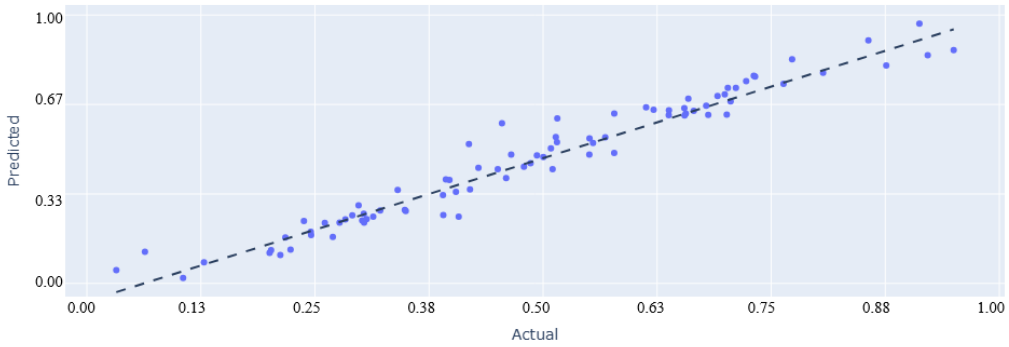

Figure 6.1: Actual vs. predicted (normalized) values - Ridge (dataset A).

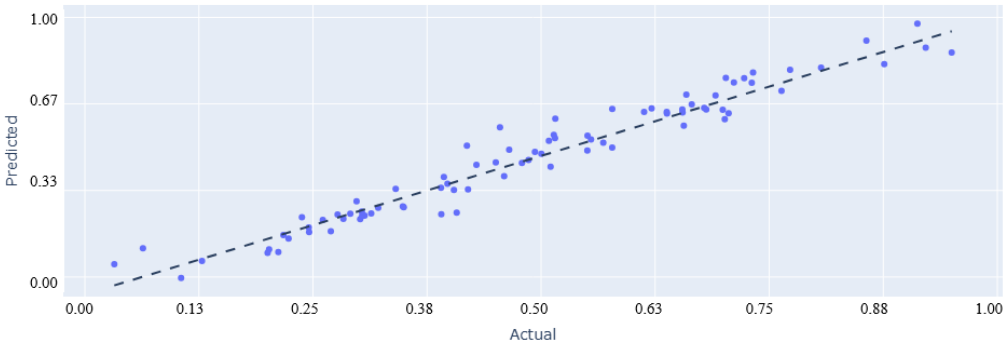

Figure 6.2: Actual vs. predicted values - Polynomial Ridge (dataset A).

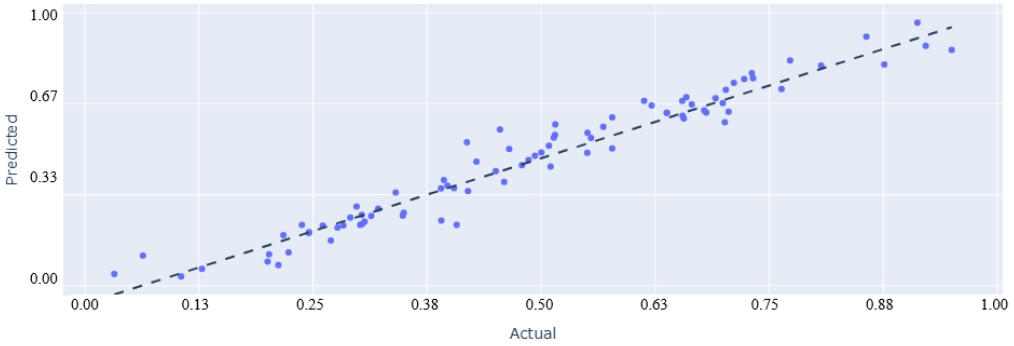

Figure 6.3: Actual vs. predicted (normalized) values - tabPFN (dataset A).

These results show that all three models achieve strong predictive performance with R^2 values consistently above 0.94, while nRMSE and nMAE remain low. The lightweight models deliver competitive results, with minimal differences in error spread (Err-STD). TabPFN, despite its higher model complexity, does not significantly outperform the simpler baselines. This suggests that the linear or polynomial models, when combined with appropriate preprocessing, are already sufficient to capture underlying patterns in the target dataset. Consequently, in this specific context, the use of complex transformer-based models might not provide real benefits, especially considering their higher computational cost. In contrast,

lightweight linear models remain as the main candidates for subsequent incremental learning experiments.

6.2 Multi-Dataset Experiments

In this section, the objective is to investigate the potential benefits of combining multiple datasets. The following experiments evaluate model performance on various combinations of the target dataset A and older-generation datasets, considering both full-dataset and incremental learning scenarios. The Ridge and Polynomial Ridge models are trained according to the pipelines described in the baseline section 6.1.1. The analysis consists of:

1. Assessing baseline performance when models are trained on the entire combined datasets under two specific configurations (described below in subsection 6.2.1).

2. Analyzing the baseline reference models (Ridge and Polynomial Ridge trained on dataset A only) (see subsection 6.2.2). The subsequent results will be compared to the Ridge model.

3. Examining how the models' predictive performance evolves as training samples are gradually incorporated (see subsection 6.2.3). This evaluation is particularly important to assess model behavior when only a limited amount of data is available.

4. Exploring briefly the similarities and relationships between datasets (see subsection 6.2.4)

6.2.1 Full-Dataset Training on Combined Datasets

Initially, the models are trained on the full combined dataset using a global `StandardScaler`, omitting product-specific information in the features. Subsequently, experiments are repeated using per-product scaling and introducing binary labels to indicate the origin of each sample during training. These binary labels help the model to explicitly account for dataset differences.

Sample weights (different for each experiment, but always in favor of dataset A) are also applied to give higher importance to dataset A samples relative to older datasets. Train-test splits are stratified by `LOT_Waf` with a fixed random state of 42 for reproducibility. The test set size is 20% of the original combined dataset.

Metrics are computed both globally and per product group, and scatter plots (similarly to baseline experiments) are provided for visual inspection of model performance. The results of these experiments are presented below, grouped by combination.

Moreover, after evaluating pair-wise combinations of the target dataset A with the older-generation datasets, further experiments include a limited number of three-dataset combinations (A+B+C and A+B+D) and the full four dataset combination. The decision not to test every possible three-dataset combinations was motivated by the observed trends in the pair-wise experiments, with the result being that only the combinations producing relevant results underwent further testing, in order to limit the already large number of experimental setups.

Dataset Combination: A + B

1. Global `StandardScaler`, no binary labels in training
   Sample weight values : 1.0 for A, 0.001 for B

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+B	0.9567	1.85%	2.43%	1.83%	15.69
Poly Ridge	A+B	0.9652	1.71%	2.18%	1.73%	14.32
Ridge	A	0.9167	1.49%	1.81%	1.49%	9.33
Poly Ridge	A	0.9256	1.43%	1.70%	1.42%	8.80
Ridge	B	0.6864	1.95%	2.49%	1.90%	16.74
Poly Ridge	B	0.7504	1.78%	2.22%	1.79%	15.34

Table 6.2: Normalized performance metrics (models trained on the full combined dataset A + B, global StandardScaler, no binary labels in training).

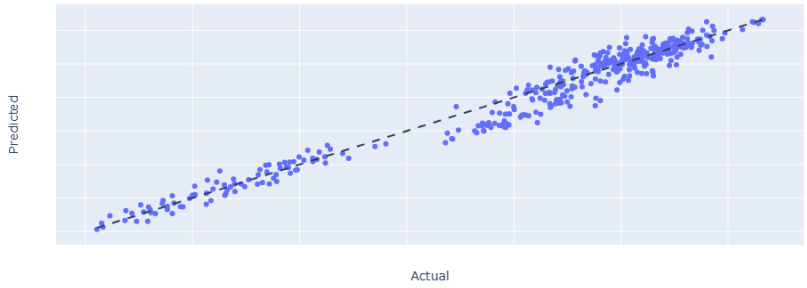

Figure 6.4: Actual vs. predicted values - Ridge (dataset A + B, global StandardScaler, no binary labels in training).

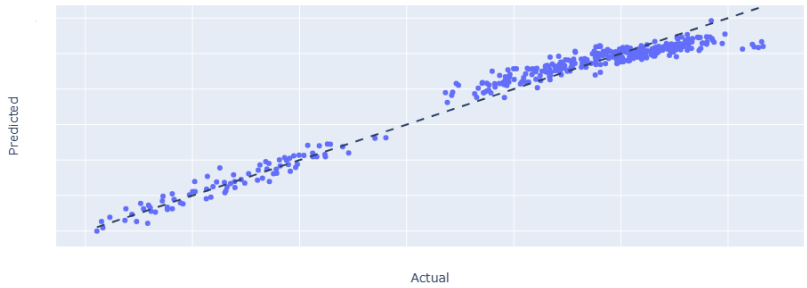

Figure 6.5: Actual vs. predicted values - Polynomial Ridge (dataset A + B, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training
   Sample weight values : 1.0 for A, 0.01 for B

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+B	0.9795	1.35%	1.67%	1.33%	10.96
Poly Ridge	A+B	0.9753	1.43%	1.84%	1.42%	11.79
Ridge	A	0.9225	1.43%	1.74%	1.43%	8.99
Poly Ridge	A	0.9242	1.42%	1.72%	1.41%	8.92
Ridge	B	0.8622	1.32%	1.65%	1.32%	11.33
Poly Ridge	B	0.8287	1.43%	1.84%	1.43%	12.26

Table 6.3: Normalized performance metrics (models trained on the full combined dataset A + B, per-product scaler, binary labels in training).

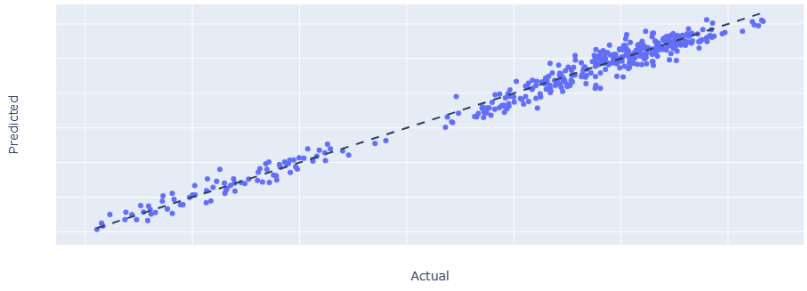

Figure 6.6: Actual vs. predicted values - Ridge (dataset A + B, per-product scaler, binary labels in training).

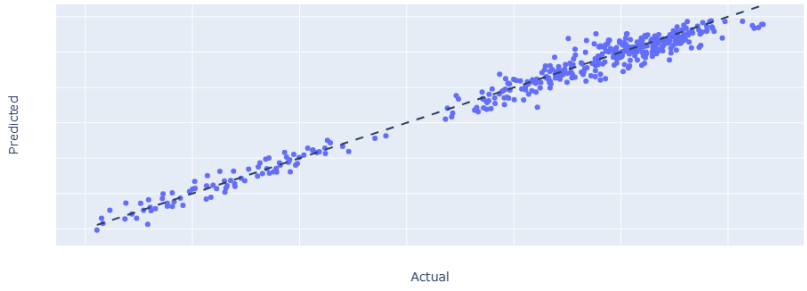

Figure 6.7: Actual vs. predicted values - Polynomial Ridge (dataset A + B, per-product scaler, binary labels in training).

Dataset Combination: A + C

1. Global `StandardScaler`, no binary labels in training
   Sample weight values : 1.0 for A, 0.01 for C

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+C	0.8499	1.48%	2.24%	1.46%	11.31
Poly Ridge	A+C	0.8350	1.60%	2.34%	1.57%	11.83
Ridge	A	0.9484	1.18%	1.58%	1.18%	8.19
Poly Ridge	A	0.9458	1.29%	1.62%	1.28%	8.39
Ridge	C	0.7631	1.58%	2.43%	1.55%	12.15
Poly Ridge	C	0.7379	1.70%	2.55%	1.67%	12.75

Table 6.4: Normalized performance metrics (models trained on the full combined dataset A + C, global StandardScaler, no binary labels in training).

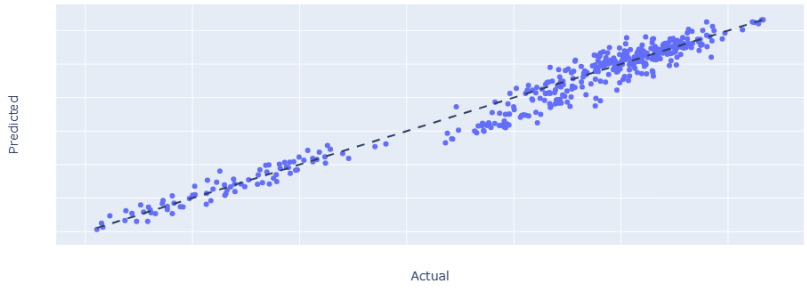

Figure 6.8: Actual vs. predicted values - Ridge (dataset A + C, global StandardScaler, no binary labels in training).

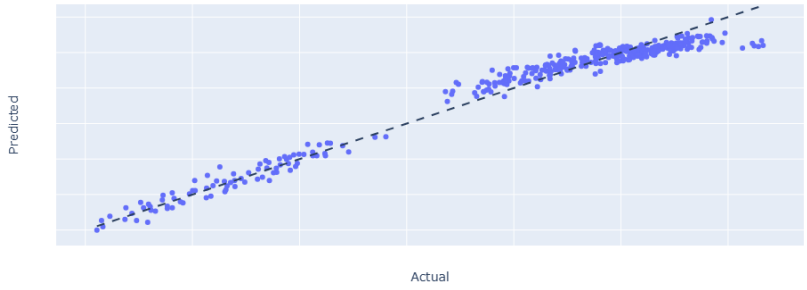

Figure 6.9: Actual vs. predicted values - Polynomial Ridge (dataset A + C, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training
   Sample weight values : 1.0 for A, 0.01 for C

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+C	0.7441	2.08%	2.92%	2.05%	14.15
Poly Ridge	A+C	0.8120	1.76%	2.50%	1.71%	12.19
Ridge	A	0.9516	1.13%	1.53%	1.13%	7.93
Poly Ridge	A	0.9496	1.21%	1.56%	1.20%	8.09
Ridge	C	0.5692	2.39%	3.27%	2.36%	15.39
Poly Ridge	C	0.6937	1.93%	2.76%	1.88%	13.02

Table 6.5: Normalized performance metrics (models trained on the full combined dataset A + C, per-product scaler, binary labels in training).

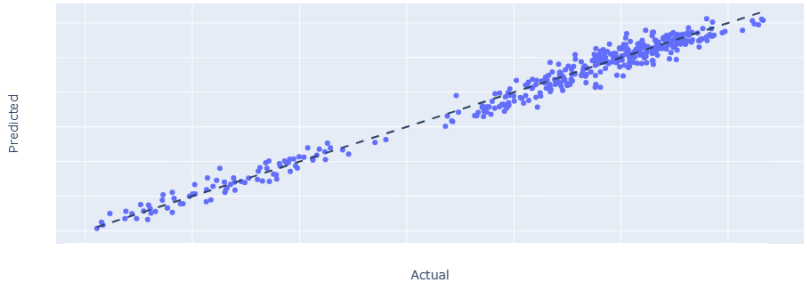

Figure 6.10: Actual vs. predicted values - Ridge (dataset A + C, per-product scaler, binary labels in training).

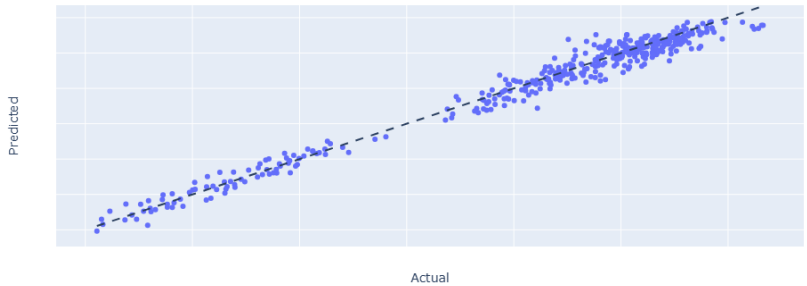

Figure 6.11: Actual vs. predicted values - Polynomial Ridge (dataset A + C, per-product scaler, binary labels in training).

Dataset Combination: A + D

1. Global `StandardScaler`, no binary labels in training
   Sample weight values : 1.0 for A, 0.01 for D

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+D	0.9770	1.55%	2.02%	1.54%	12.97
Poly Ridge	A+D	0.9878	1.18%	1.47%	1.16%	9.42
Ridge	A	0.9323	1.36%	1.72%	1.37%	9.01
Poly Ridge	A	0.9327	1.38%	1.72%	1.38%	8.98
Ridge	D	0.8095	1.62%	2.06%	1.59%	14.24
Poly Ridge	D	0.9135	1.10%	1.39%	1.10%	9.57

Table 6.6: Normalized performance metrics (models trained on the full combined dataset A + D, global StandardScaler, no binary labels in training).

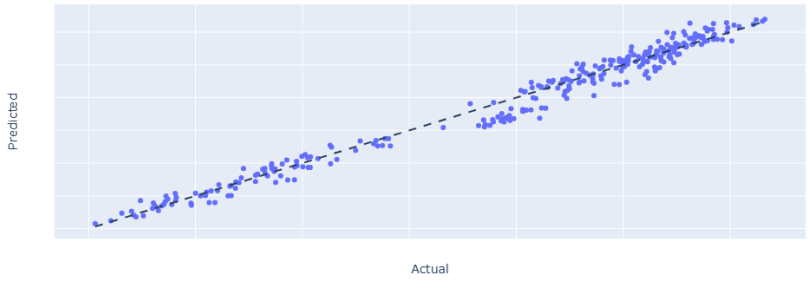

Figure 6.12: Actual vs. predicted values - Ridge (dataset A + D, global StandardScaler, no binary labels in training).

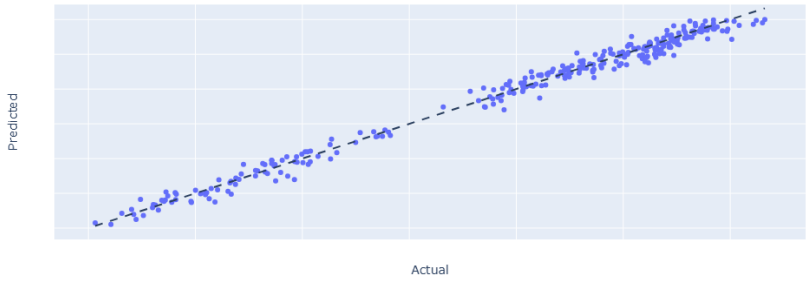

Figure 6.13: Actual vs. predicted values - Polynomial Ridge (dataset A + D, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training
   Sample weight values : 1.0 for A, 0.01 for D

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+D	0.9848	1.33%	1.64%	1.32%	10.35
Poly Ridge	A+D	0.9239	2.66%	3.67%	2.76%	21.29
Ridge	A	0.9278	1.37%	1.78%	1.37%	9.31
Poly Ridge	A	0.9308	1.40%	1.74%	1.40%	9.11
Ridge	D	0.8863	1.32%	1.59%	1.31%	10.61
Poly Ridge	D	0.3007	3.17%	3.95%	3.18%	23.16

Table 6.7: Normalized performance metrics (models trained on the full combined dataset A + D, per-product scaler, binary labels in training).

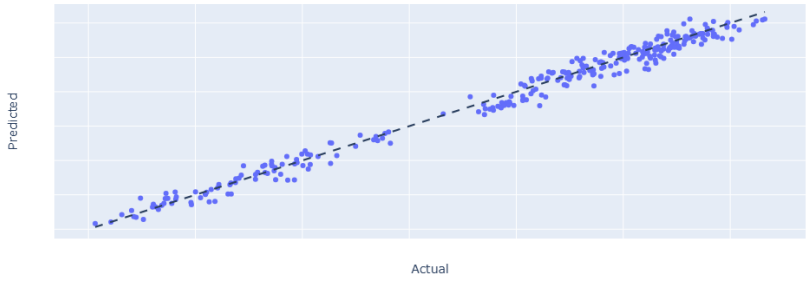

Figure 6.14: Actual vs. predicted values - Ridge (dataset A + D, per-product scaler, binary labels in training).

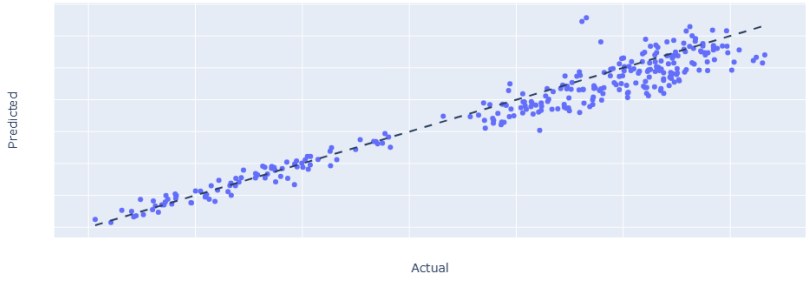

Figure 6.15: Actual vs. predicted values - Polynomial Ridge (dataset A + D, per-product scaler, binary labels in training).

Dataset Combination: A + B + C

1. Global `StandardScaler`, no binary labels in training
   Sample weight values : 1.0 for A, 0.01 for B and C

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+B+C	0.9835	1.51%	2.12%	1.47%	12.69
Poly Ridge	A+B+C	0.9875	1.33%	1.85%	1.27%	11.06
Ridge	A	0.9352	1.27%	1.61%	1.27%	8.34
Poly Ridge	A	0.9309	1.37%	1.66%	1.36%	8.58
Ridge	B	0.8493	1.42%	1.79%	1.40%	12.37
Poly Ridge	B	0.9162	1.04%	1.33%	1.04%	9.24
Ridge	C	0.6950	1.70%	2.82%	1.65%	14.14
Poly Ridge	C	0.7163	1.68%	2.72%	1.64%	13.65

Table 6.8: Normalized performance metrics (models trained on the full combined dataset A + B + C, global StandardScaler, no binary labels in training).

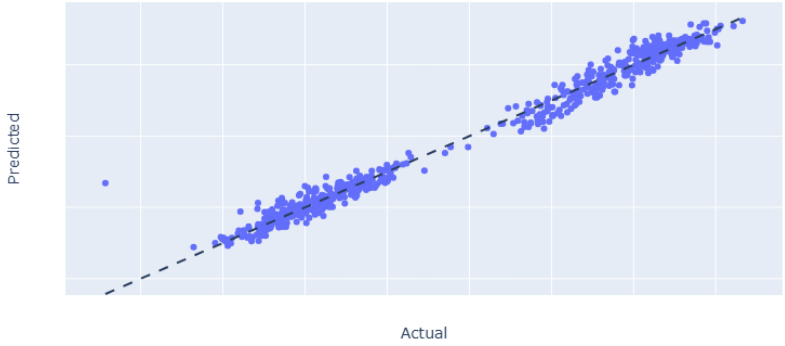

Figure 6.16: Actual vs. predicted values - Ridge (dataset A + B + C, global StandardScaler, no binary labels in training).

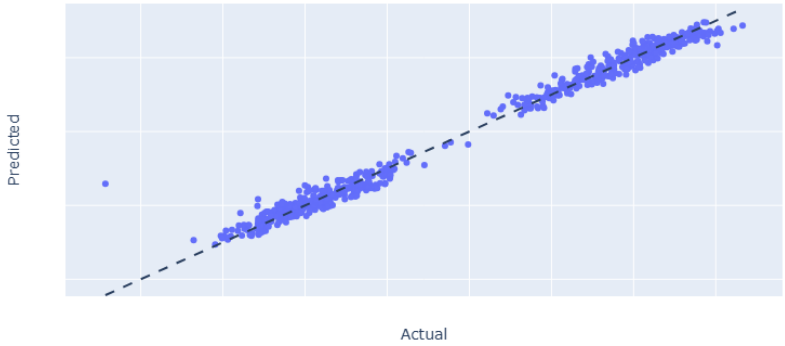

Figure 6.17: Actual vs. predicted values - Polynomial Ridge (dataset A + B + C, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training
   Sample weight values : 1.0 for A, 0.01 for B and C

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+B+C	0.9818	1.67%	2.22%	1.59%	13.33
Poly Ridge	A+B+C	0.9463	3.02%	3.82%	3.00%	22.69
Ridge	A	0.9364	1.26%	1.59%	1.26%	8.24
Poly Ridge	A	0.9370	1.25%	1.59%	1.25%	8.18
Ridge	B	0.8779	1.30%	1.61%	1.30%	11.11
Poly Ridge	B	0.3764	3.02%	3.63%	3.02%	20.19
Ridge	C	0.5692	2.29%	3.35%	2.22%	16.80
Poly Ridge	C	0.1967	3.61%	4.57%	3.55%	19.97

Table 6.9: Normalized performance metrics (models trained on the full combined dataset A + B + C, per-product scaler, binary labels in training).

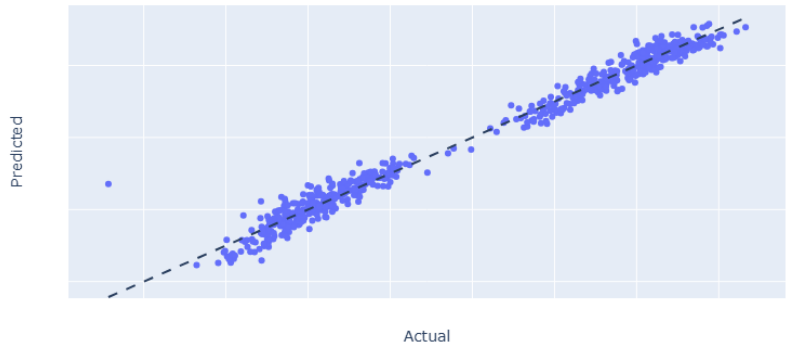

Figure 6.18: Actual vs. predicted values - Ridge (dataset A + B + C, per-product scaler, binary labels in training).

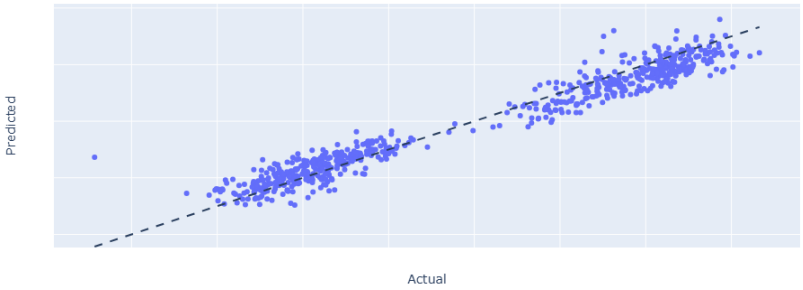

Figure 6.19: Actual vs. predicted values - Polynomial Ridge (dataset A + B + C, per-product scaler, binary labels in training).

Dataset Combination: A + B + D

1. Global `StandardScaler`, no binary labels in training
   Sample weight values : 1.0 for A, 0.01 for B and D

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+B+D	0.9700	1.37%	1.73%	1.36%	11.61
Poly Ridge	A+B+D	0.9789	1.13%	1.45%	1.12%	9.73
Ridge	A	0.9351	1.36%	1.64%	1.34%	8.53
Poly Ridge	A	0.9377	1.31%	1.61%	1.29%	8.35
Ridge	B	0.8467	1.42%	1.79%	1.41%	12.32
Poly Ridge	B	0.8975	1.13%	1.46%	1.12%	10.03
Ridge	D	0.8754	1.30%	1.62%	1.28%	11.06
Poly Ridge	D	0.9114	1.06%	1.37%	1.06%	9.45

Table 6.10: Normalized performance metrics (models trained on the full combined dataset A + B + D, global StandardScaler, no binary labels in training).

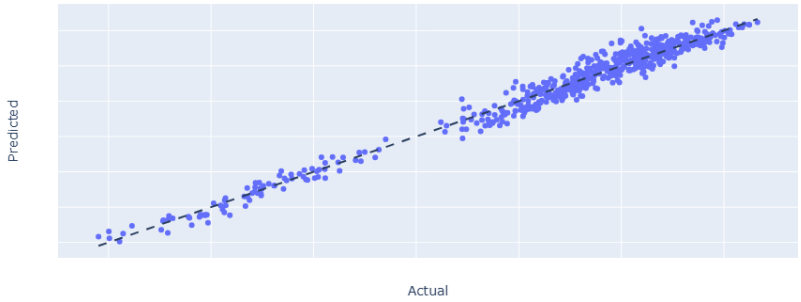

Figure 6.20: Actual vs. predicted values - Ridge (dataset A + B + D, global StandardScaler, no binary labels in training).

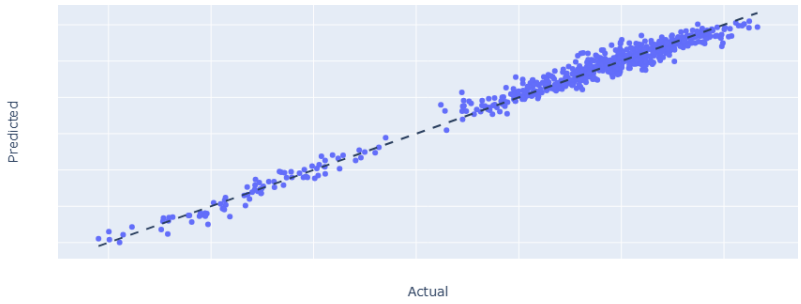

Figure 6.21: Actual vs. predicted values - Polynomial Ridge (dataset A + B + D, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training
   Sample weight values : 1.0 for A, 0.01 for B and D

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+B+D	0.9759	1.25%	1.55%	1.24%	10.41
Poly Ridge	A+B+D	0.9788	1.18%	1.45%	1.17%	9.75
Ridge	A	0.9430	1.27%	1.54%	1.26%	8.01
Poly Ridge	A	0.9379	1.33%	1.61%	1.32%	8.37
Ridge	B	0.8764	1.30%	1.61%	1.30%	11.13
Poly Ridge	B	0.8985	1.18%	1.46%	1.18%	10.02
Ridge	D	0.9024	1.16%	1.44%	1.15%	9.86
Poly Ridge	D	0.9086	1.11%	1.39%	1.10%	9.23

Table 6.11: Normalized performance metrics (models trained on the full combined dataset A + B + D, per-product scaler, binary labels in training).

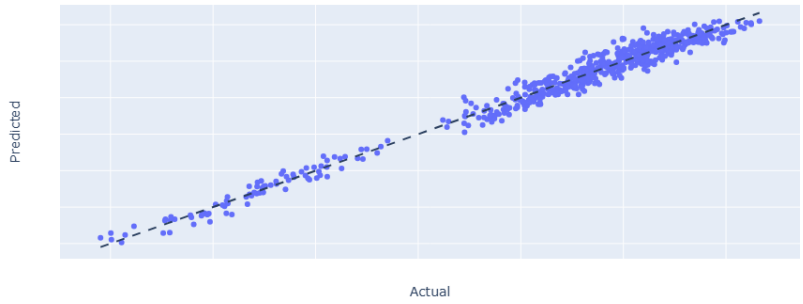

Figure 6.22: Actual vs. predicted values - Ridge (dataset A + B + D, per-product scaler, binary labels in training).

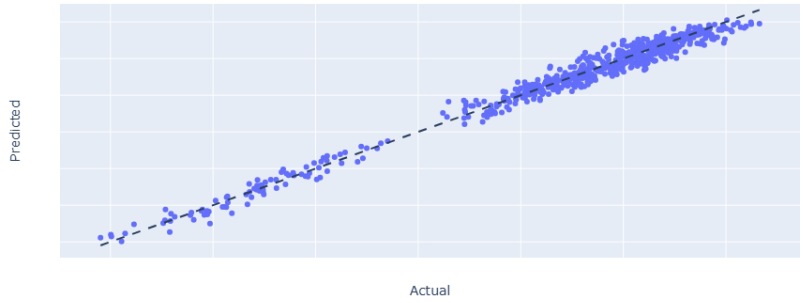

Figure 6.23: Actual vs. predicted values - Polynomial Ridge (dataset A + B + D, per-product scaler, binary labels in training).

Dataset Combination: A + B + C + D

1. Global `StandardScaler`, no binary labels in training
   Sample weight values : 1.0 for A, 0.001 for B, C and D

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+B+C+D	0.9755	1.81%	2.44%	1.81%	15.15
Poly Ridge	A+B+C+D	0.9830	1.52%	2.04%	1.52%	12.68
Ridge	A	0.9477	1.13%	1.47%	1.13%	7.53
Poly Ridge	A	0.9407	1.22%	1.57%	1.22%	8.07
Ridge	B	0.7487	1.84%	2.28%	1.82%	15.43
Poly Ridge	B	0.8695	1.29%	1.64%	1.29%	11.39
Ridge	C	0.6323	2.15%	3.05%	2.11%	13.65
Poly Ridge	C	0.6408	2.19%	3.01%	2.15%	15.03
Ridge	D	0.7645	1.79%	2.33%	1.75%	14.82
Poly Ridge	D	0.8527	1.45%	1.85%	1.44%	12.64

Table 6.12: Normalized performance metrics (models trained on the full combined dataset A + B + C + D, global StandardScaler, no binary labels in training).

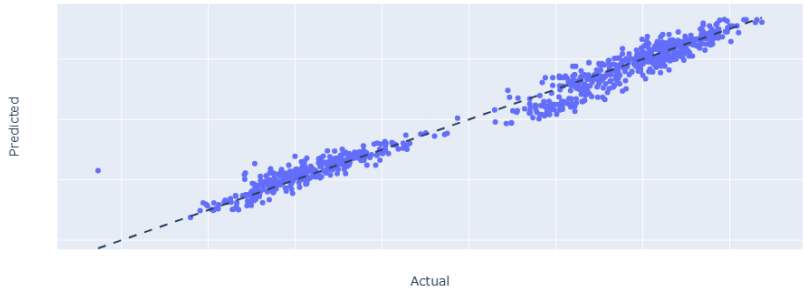

Figure 6.24: Actual vs. predicted values - Ridge (dataset A + B + C + D, global StandardScaler, no binary labels in training).

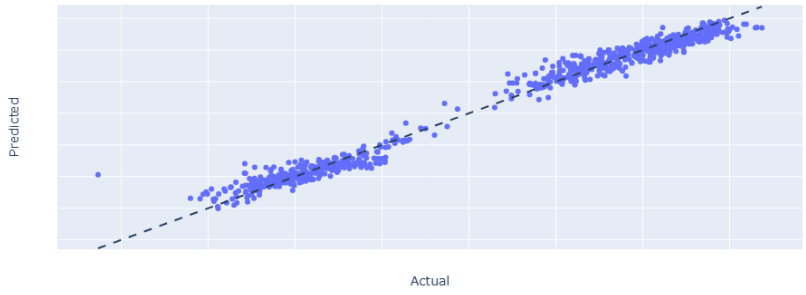

Figure 6.25: Actual vs. predicted values - Polynomial Ridge (dataset A + B + C + D, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training
   Sample weight values : 1.0 for A, 0.001 for B, C and D

Algorithm	Test Set	R^2	MAPE	nRMSE	nMAE	Err-STD
Ridge	A+B+C+D	0.9764	1.92%	2.39%	1.83%	14.89
Poly Ridge	A+B+C+D	0.9591	2.53%	3.15%	2.46%	19.61
Ridge	A	0.9495	1.08%	1.45%	1.07%	7.35
Poly Ridge	A	0.9390	1.19%	1.59%	1.20%	8.14
Ridge	B	0.7871	1.74%	2.10%	1.74%	14.55
Poly Ridge	B	0.6490	2.12%	2.69%	2.12%	17.77
Ridge	C	0.4648	2.85%	3.68%	2.77%	18.42
Poly Ridge	C	0.1755	3.63%	4.57%	3.57%	19.41
Ridge	D	0.8562	1.43%	1.82%	1.42%	12.55
Poly Ridge	D	0.6433	2.40%	2.87%	2.39%	17.79

Table 6.13: Normalized performance metrics (models trained on the full combined dataset A + B + C + D, per-product scaler, binary labels in training).

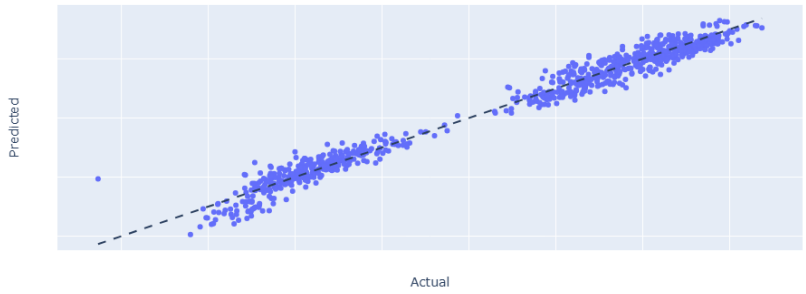

Figure 6.26: Actual vs. predicted values - Ridge (dataset A + B + C + D, per-product scaler, binary labels in training).

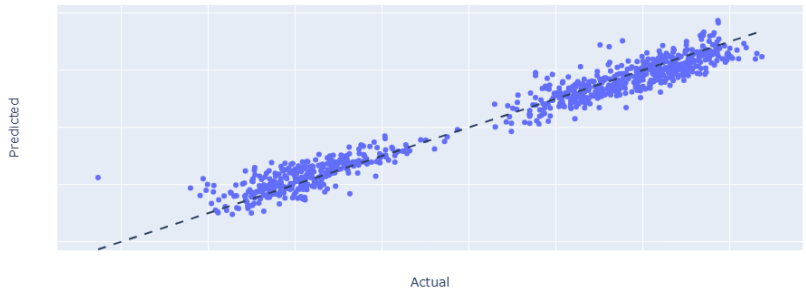

Figure 6.27: Actual vs. predicted values - Polynomial Ridge (dataset A + B + C + D, per-product scaler, binary labels in training).

6.2.2 Dataset A Learning Curves (Baseline Reference)

These learning curves are generated with active learning. They provide a baseline reference that will be used to compare the results obtained in the subsequent experiments, while assessing the behavior of the models under increasing training data (trained only on dataset A, incrementally enlarging the training set size). The analysis focuses on the normalized root mean squared error (nRMSE) and the error standard deviation (err-STD) as evaluation metrics. The plots below (figs. 6.28 and 6.29) describe those metrics versus the training set size.

Figure 6.28: Learning curves of Ridge and Polynomial Ridge models on dataset A: nRMSE versus training set size.

Figure 6.29: Learning curves of Ridge and Polynomial Ridge models on dataset A: err-STD versus training set size.

Model	Metric	32	59	98	330
Ridge	nRMSE	1.86%	1.54%	1.52%	1.53%
	err-STD	9.60	7.91	7.84	7.91
Polynomial Ridge	nRMSE	2.21%	1.79%	1.59%	1.55%
	err-STD	11.47	9.26	8.22	8.03

Table 6.14: Comparison of nRMSE and err-STD at specific training set sizes for Ridge and Polynomial Ridge models (corresponding to learning curves in figs. 6.28 and 6.29).

As demonstrated by these results (where 330 corresponds to the point where all available samples are used), Ridge model outperforms the polynomial counterpart by achieving better accuracy at the same training set size.

6.2.3 Incremental Learning Strategy Comparison

This section explores the results of the multi-dataset experiments performed with an incremental approach (as explained in section 5.3). Strategy 1 experiments are performed with active learning, while Strategy 2 experiments are performed without active learning (which will be further explored in section 6.3). It is reminded to the reader that the plots show the nRMSE and err-STD metrics versus the increasing amount of dataset A samples used in training.

Since the full multi-dataset training experiments (6.2.1) produced the best results with Ridge model, rather than the Polynomial counterpart, only the results of the Ridge model are presented for conciseness.

Strategy 1 Learning Curves (with Active Learning)

The learning curves obtained with this strategy turned out to be highly irregular, exhibiting strong oscillation and noise. This was not conducive to the formulation of any meaningful interpretation of the model's behavior as training data increased. For this reason, only a few symbolic examples are reported here to illustrate the instability of this approach, which ultimately proved unfeasible for the analysis. Moreover, in order to make the figures more interpretable, only the baseline reference Ridge on dataset A and Strategy 1 Ridge learning curves are shown for comparison.

- **Dataset Combination: A + B**
  Per-product scaler, binary labels in training
  Sample weights values : 1.0 for A, 0.00001 for B

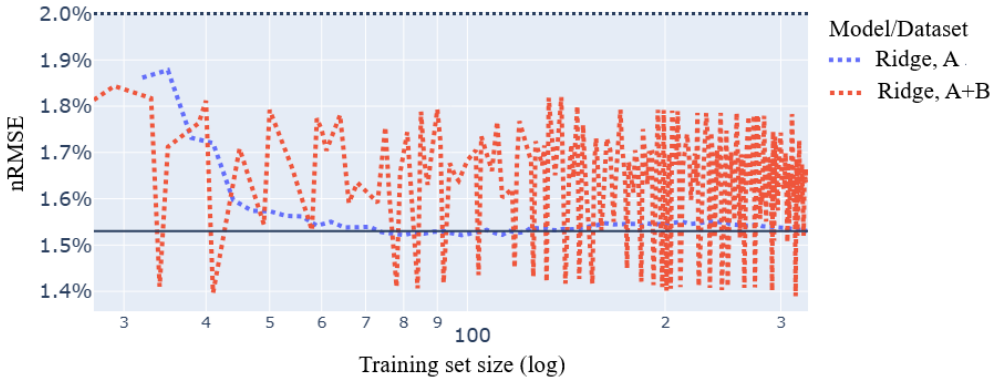

Figure 6.30: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 1 (A+B).

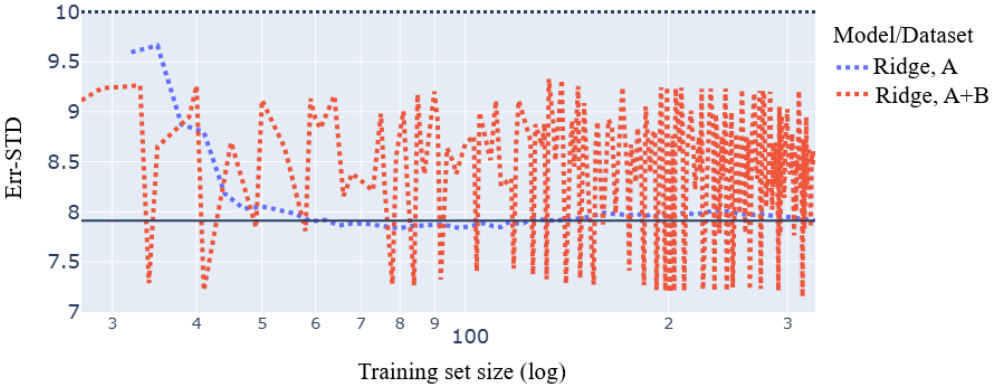

Figure 6.31: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 1 (A+B).

- **Dataset Combination: A + B + C**
  Per-product scaler, binary labels in training
  Sample weights values : 1.0 for A, 0.00001 for B and C

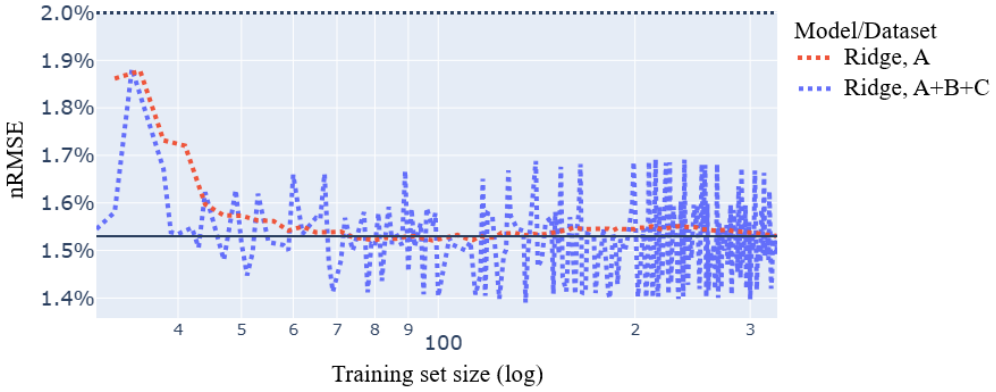

Figure 6.32: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 1 (A+B+C).

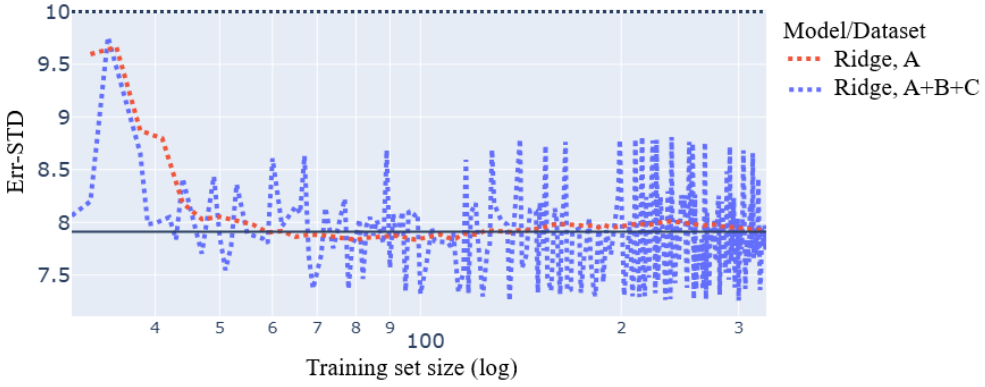

Figure 6.33: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 1 (A+B+C).

- **Dataset Combination: A + B + C + D**
  Per-product scaler, binary labels in training
  Sample weights values : 1.0 for A, 0.00001 for B,C and D

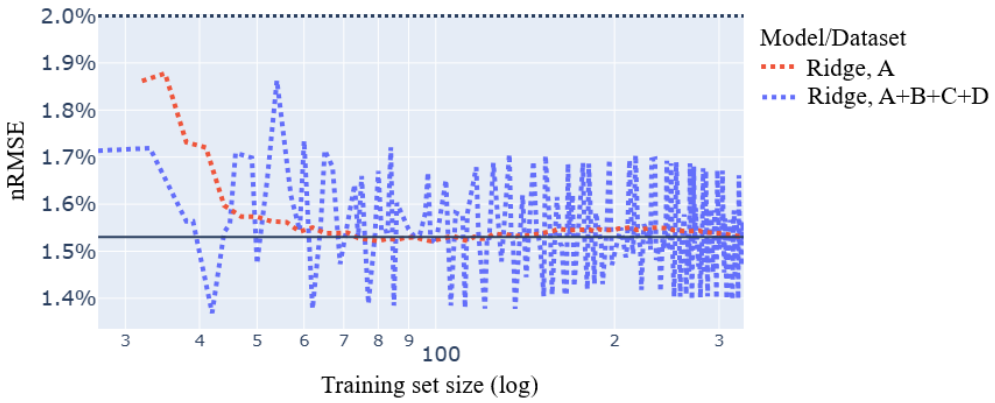

Figure 6.34: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 1 (A+B+C+D).

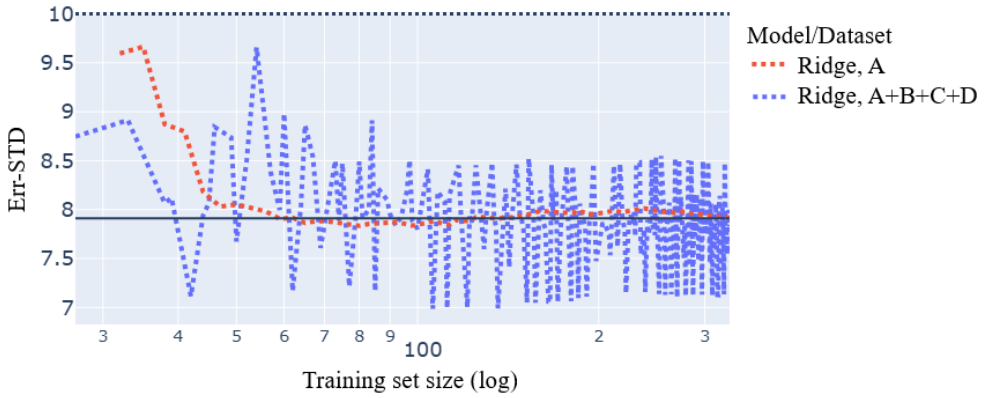

Figure 6.35: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 1 (A+B+C+D).

As shown, while Strategy 1 is highly ineffective and produces learning curves with strong oscillations and irregularities, the initial portion of the curve still reflects the accuracy advantage achievable with fewer training samples when combining datasets.

Strategy 2 Learning Curves (without Active Learning)

This strategy, unlike the previous approach, produced consistent and interpretable learning curves. Therefore, all dataset combinations already introduced in 6.2.1 are systematically analyzed here.

To assess the effect of per-product scaling and binary labels, the following results *for each combination of datasets*, are structured in this way:

1. Global `StandardScaler`, no binary labels in training
2. Per-product scaler, binary labels in training

- **Dataset Combination: A + B**
  Sample weights values : 1.0 for A, 0.00001 for B

    1. Global `StandardScaler`, no binary labels in training

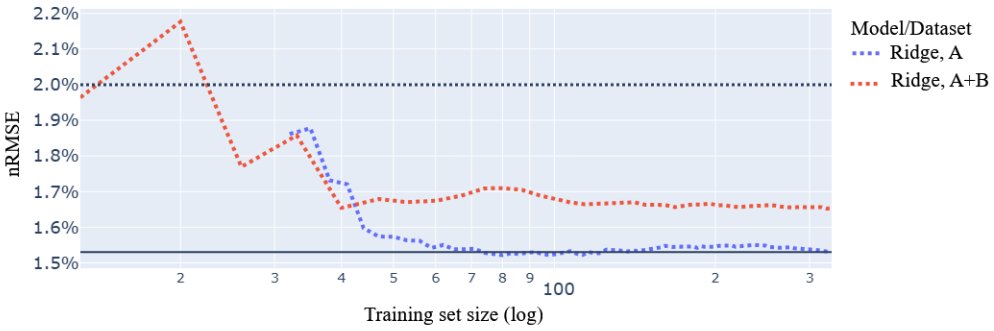

Figure 6.36: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B, global StandardScaler, no binary labels in training).

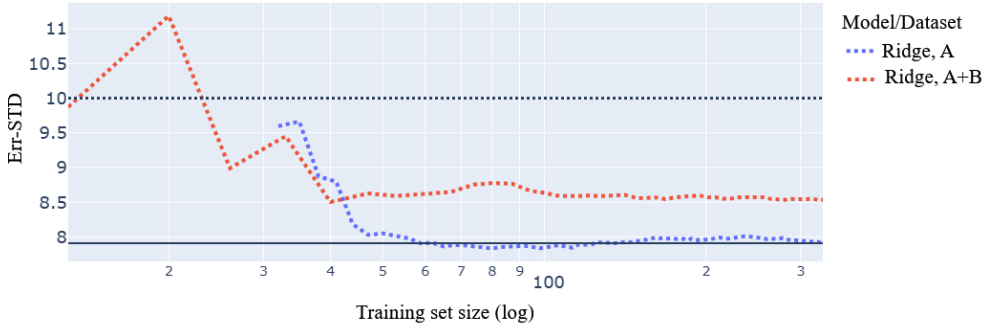

Figure 6.37: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training

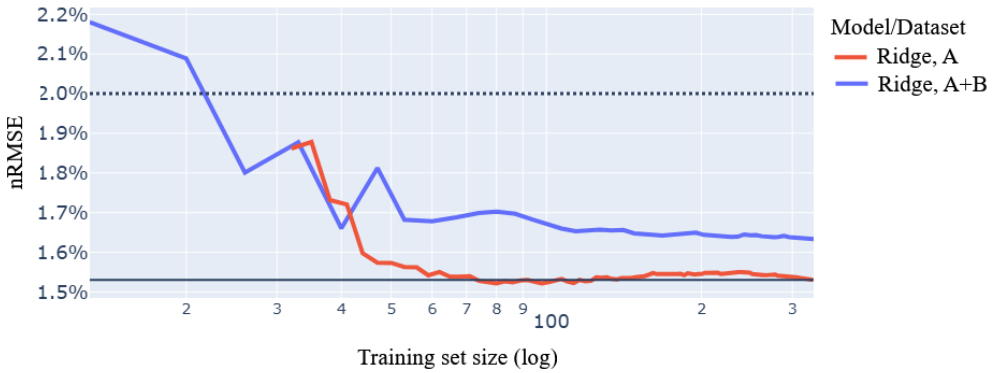

Figure 6.38: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B, per-product scaler, binary labels in training).

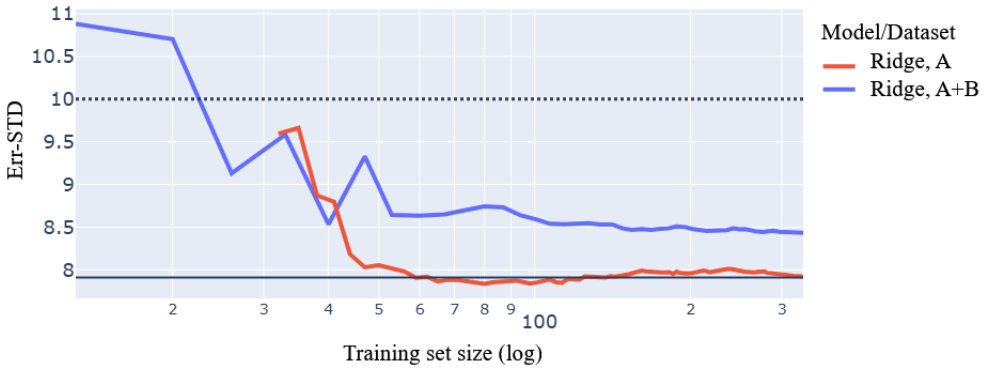

Figure 6.39: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B, per-product scaler, binary labels in training).

- **Dataset Combination: A + C**
  Sample weights values : 1.0 for A, 0.00001 for C

  1. Global `StandardScaler`, no binary labels in training

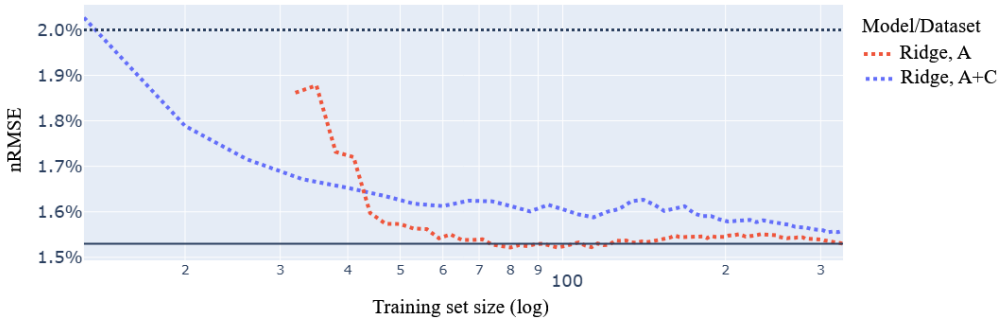

Figure 6.40: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+C, global StandardScaler, no binary labels in training).

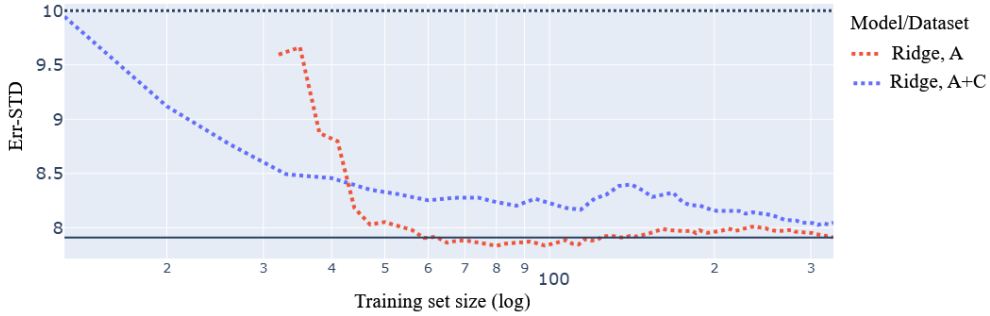

Figure 6.41: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+C, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training

Figure 6.42: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+C, per-product scaler, binary labels in training).

Figure 6.43: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+C, per-product scaler, binary labels in training).

- **Dataset Combination: A + D**
  Sample weights values : 1.0 for A, 0.00001 for D

  1. Global `StandardScaler`, no binary labels in training

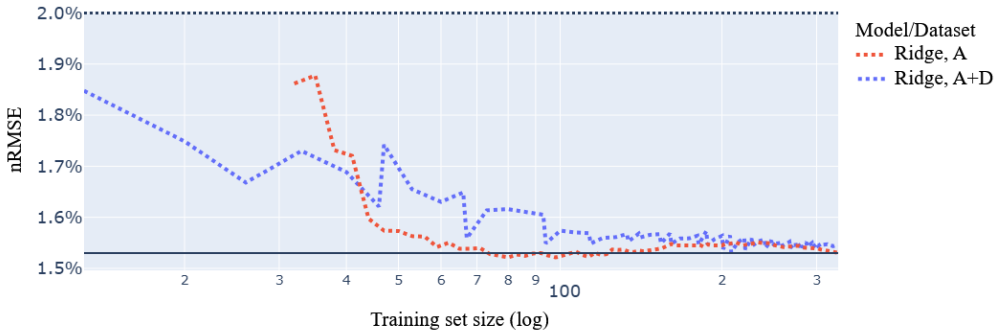

Figure 6.44: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+D, global StandardScaler, no binary labels in training).

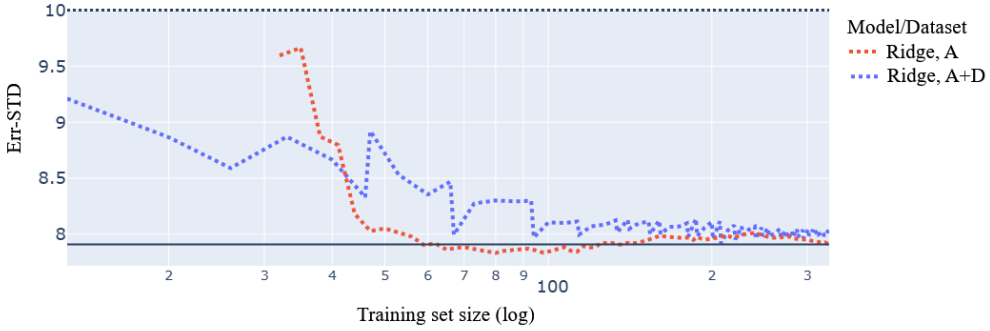

Figure 6.45: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+D, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training

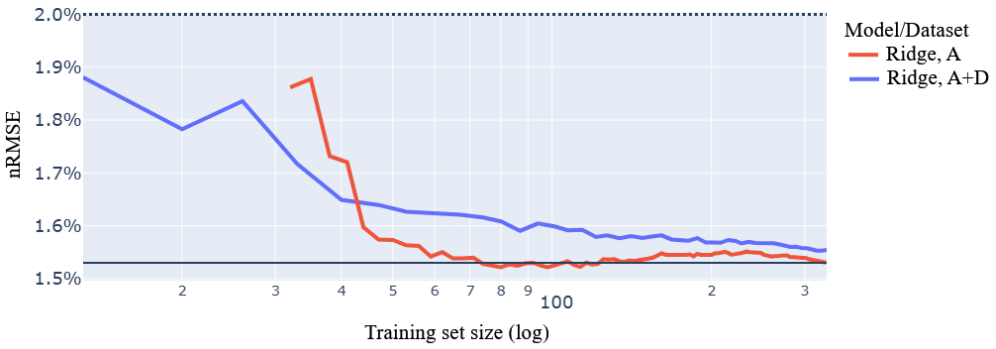

Figure 6.46: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+D, per-product scaler, binary labels in training).

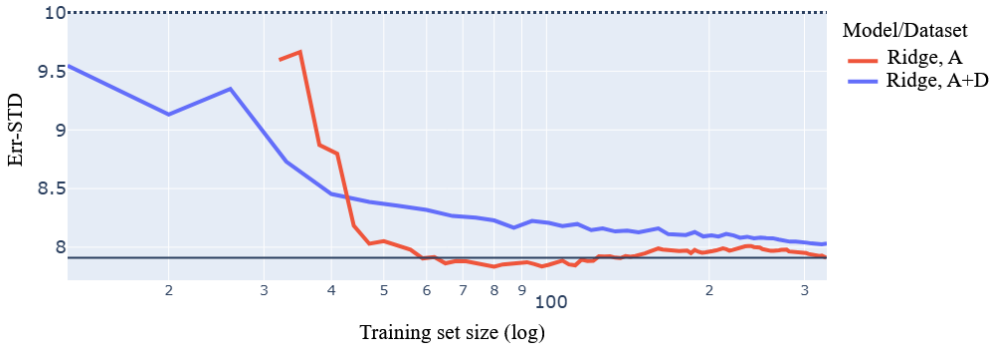

Figure 6.47: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+D, per-product scaler, binary labels in training).

- **Dataset Combination: A + B + C**
  Sample weights values : 1.0 for A, 0.00001 for B and C

  1. Global `StandardScaler`, no binary labels in training

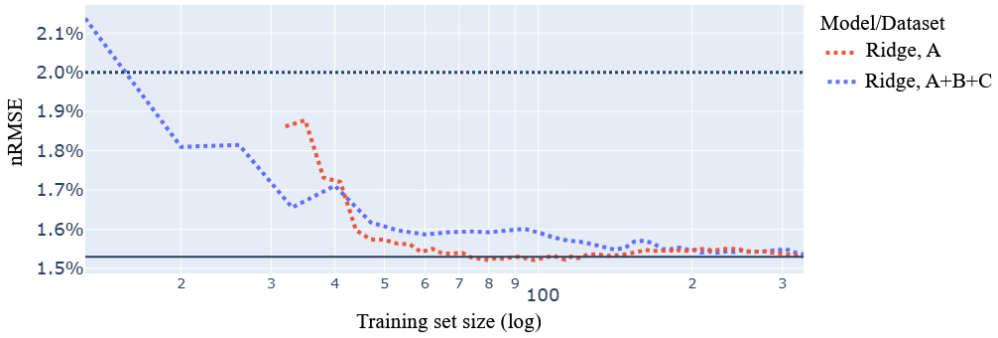

Figure 6.48: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+C, global StandardScaler, no binary labels in training).

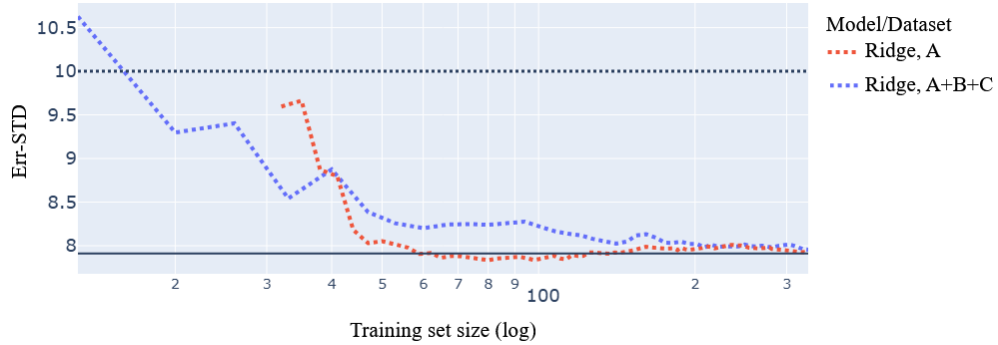

Figure 6.49: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+C, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training

Figure 6.50: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+C, per-product scaler, binary labels in training).

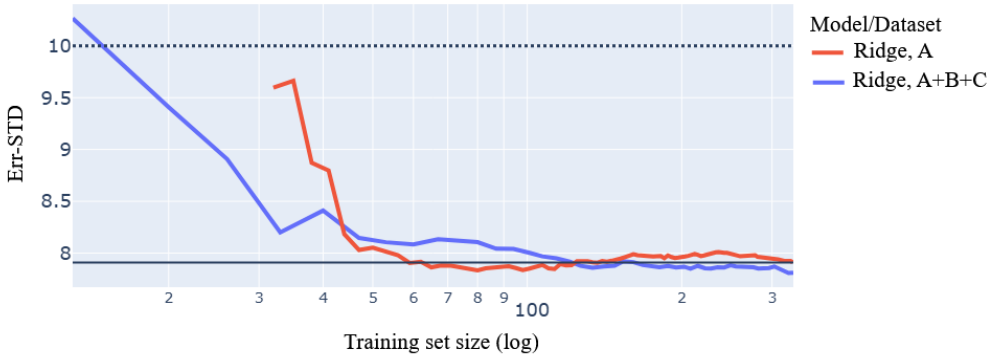

Figure 6.51: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+C, per-product scaler, binary labels in training).

- **Dataset Combination: A + B + D**
  Sample weights values : 1.0 for A, 0.00001 for B and D

  1. Global `StandardScaler`, no binary labels in training

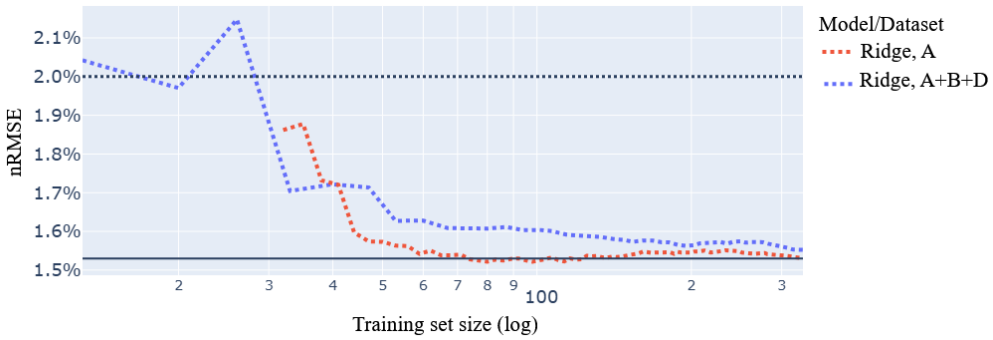

Figure 6.52: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+D, global StandardScaler, no binary labels in training).

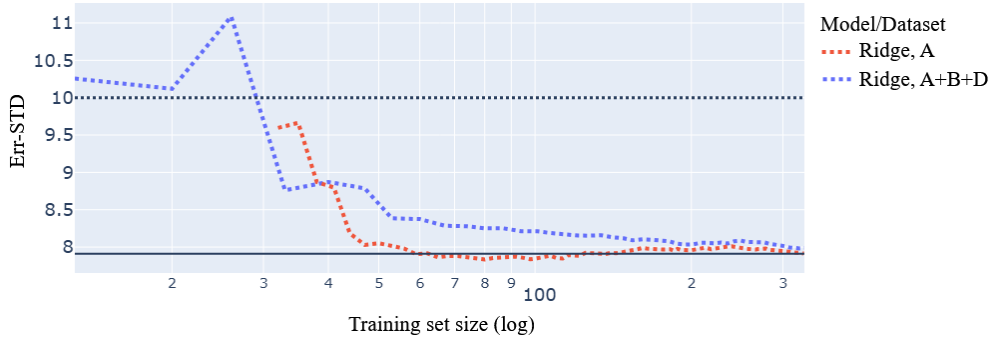

Figure 6.53: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+D, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training

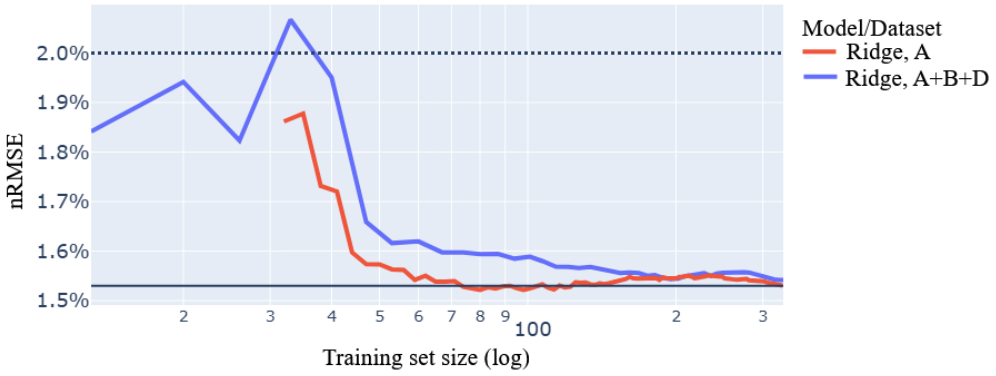

Figure 6.54: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+D, per-product scaler, binary labels in training).

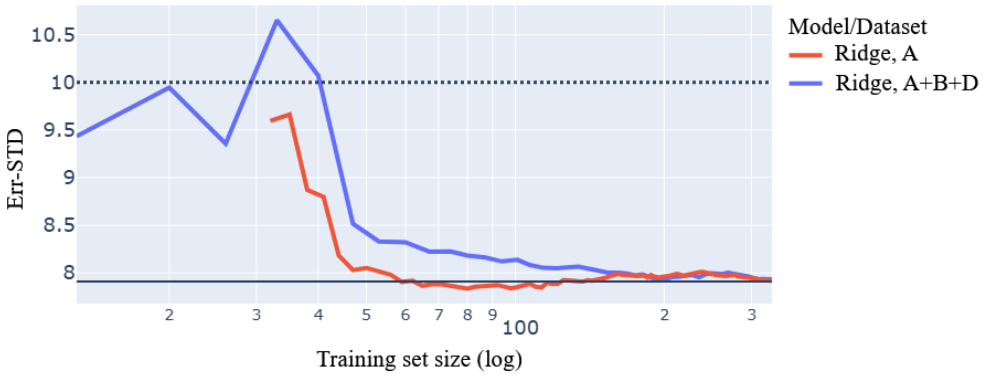

Figure 6.55: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+D, per-product scaler, binary labels in training).

- **Dataset Combination: A + B + C + D**

  Sample weights values : 1.0 for A, 0.00001 for B, C and D

  1. Global `StandardScaler`, no binary labels in training

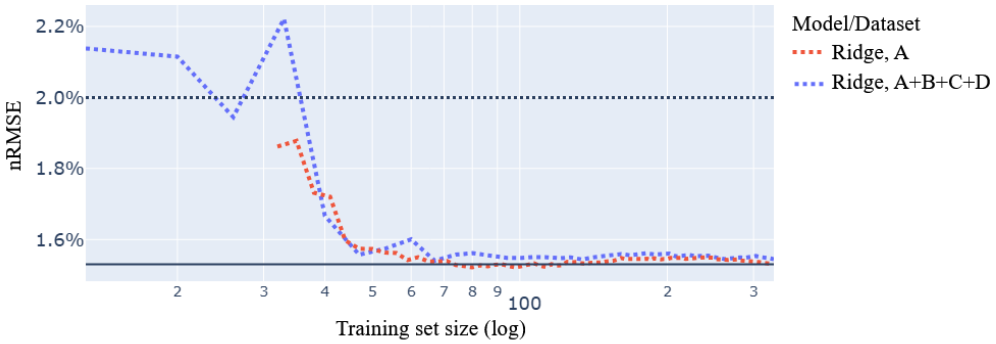

Figure 6.56: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+C+D, global StandardScaler, no binary labels in training).

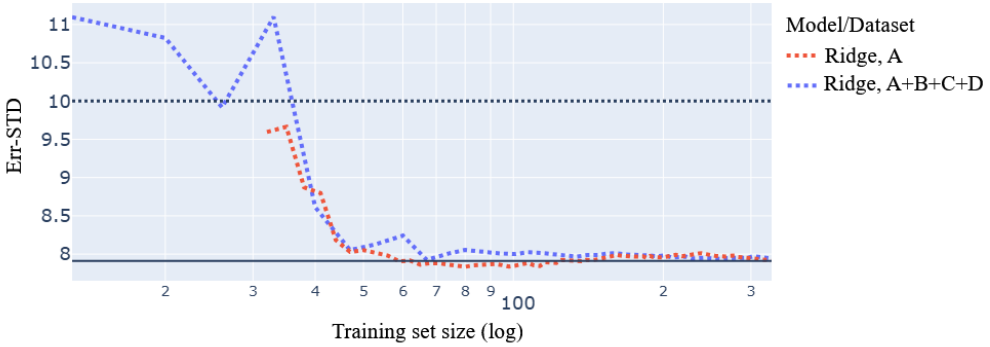

Figure 6.57: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+C+D, global StandardScaler, no binary labels in training).

2. Per-product scaler, binary labels in training

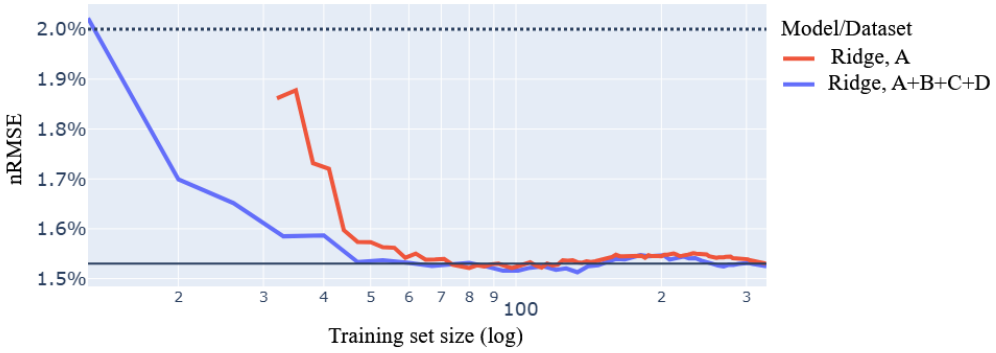

Figure 6.58: Learning curves (nRMSE versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+C+D, per-product scaler, binary labels in training).

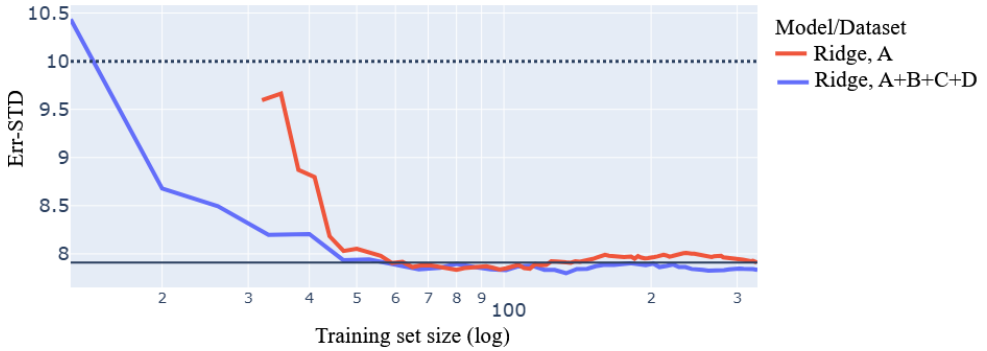

Figure 6.59: Learning curves (err-STD versus training set size): baseline Ridge model (A), Ridge with Strategy 2 (A+B+C+D, per-product scaler, binary labels in training).

Key Takeaways from Incremental Strategies Experiments

Compared to Strategy 1, in which Active Learning was applied and which proved largely ineffective and irregular, Strategy 2 (without Active Learning) demonstrates a clear advantage. Combining datasets with this approach yielded mixed outcomes when using a global StandardScaler without binary labels in training, even though in some cases (e.g. A+B+C) already showed tangible benefits over the baseline. The use of a per-product scaler with binary labels in training consistently improved stability and performance. This was clear by looking at the resulting smoother learning curves characterized by decreased error values. This configuration also produced the best results, particularly for A+B+C and A+B+C+D combinations, with the latter emerging as the most effective combination so far, achieving the lowest error with relatively small training set size. A few irregular behaviors have been observed (e.g. A+B+D or a slight increase in initial error for A+C), but they can be considered as outliers, since the overall pattern consistently points to improvements with per-product scaling. A remark has to be done for the A+C case, where it can be seen that good accuracy at few dataset A samples is already achieved without per-product scaler and binary labels (see 6.40), improving then, even further, when applying them.

These results are further highlighted through the direct curve comparison with baseline Ridge below, confirming that combination A+B+C+D with per-product scaling and binary labels represents indeed the best configuration.

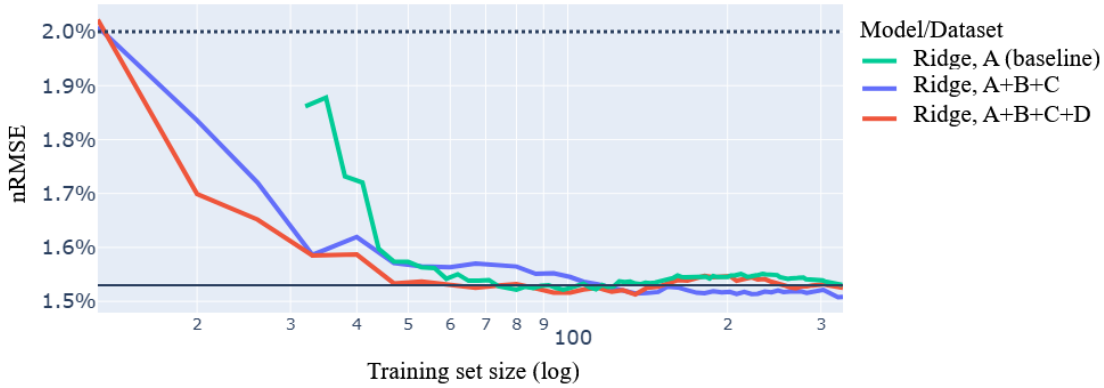

Figure 6.60: Learning curves (nRMSE versus training set size): comparison between baseline and best configurations (Strategy 2, without AL).

Model / Dataset	Training size	R^2	MAPE	nRMSE	nMAE	Err-STD
Baseline Ridge (A)	20	-	-	-	-	-
	33	0.92	1.47%	1.86%	1.48%	9.60
	50	0.94	1.22%	1.57%	1.23%	8.05
	330	0.95	1.20%	1.53%	1.20%	7.91
Ridge S2: A+B+C	20	0.93	1.50%	1.84%	1.51%	9.41
	33	0.95	1.28%	1.59%	1.28%	8.20
	50	0.95	1.28%	1.56%	1.29%	8.11
	330	0.95	1.22%	1.51%	1.22%	7.81
Ridge S2: A+B+C+D	20	0.94	1.40%	1.70%	1.40%	8.68
	33	0.95	1.27%	1.58%	1.27%	8.20
	50	0.95	1.21%	1.54%	1.21%	7.94
	330	0.95	1.19%	1.52%	1.19%	7.83

Table 6.15: Comparison of baseline Ridge (A) and the two best Strategy 2 (S2) combinations (without AL) at different training set sizes. Metrics shown: R^2, MAPE, nRMSE, nMAE, Err-STD.

6.2.4 Dataset Similarity Analysis

To assess the similarity between datasets, *Jensen-Shannon Divergence (JSD)* has been used. JSD provides a measure of difference between probability distributions. The divergence value ranges from 0 to 1, with values closer to 0 representing higher similarity in the distribution. The analysis was performed both on the feature and target variable distributions, comparing datasets B, C, and D against the reference dataset A. The results for feature analysis were grouped into ranges and visualized in a grouped bar plot, showing the percentage of features within each divergence range for each dataset. This brief analysis provides information that might help identify potential directions to further improve accuracy and provide additional insight into the results of this work.

Feature Similarity Analysis

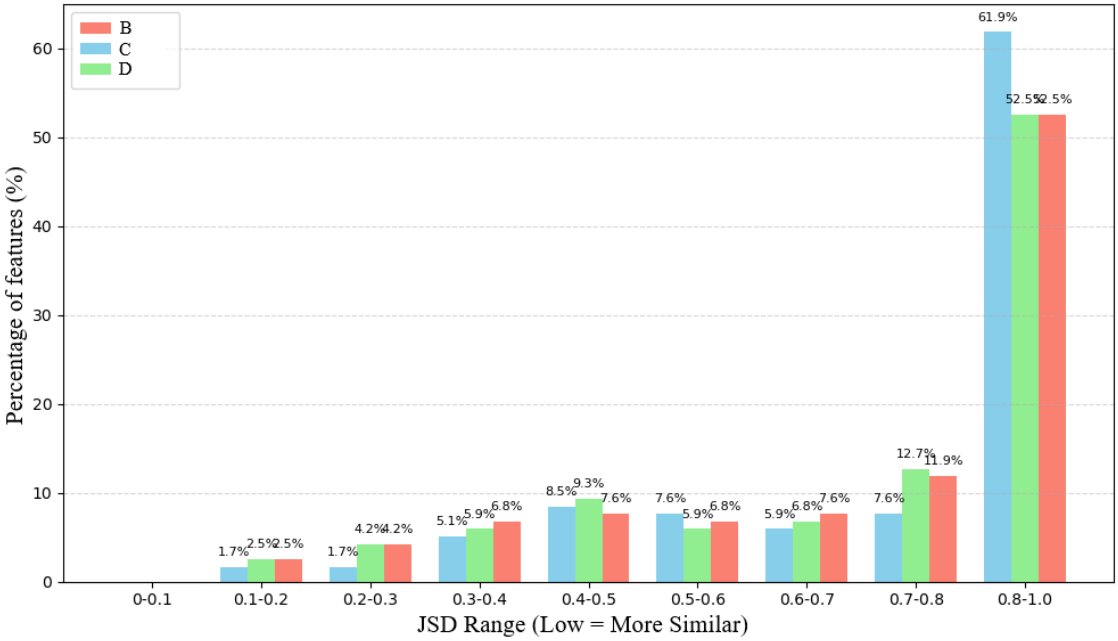

Figure 6.61: Datasets B, C, and D grouped bar plot of percentage of features per divergence ranges with respect to dataset A.

It is evident from the plot that the distributions of many features in each datasets differ substantially from their counterparts in dataset A. Nevertheless, it is important to note that each dataset also contains features with similar distributions.

Prediction Target Variable Analysis

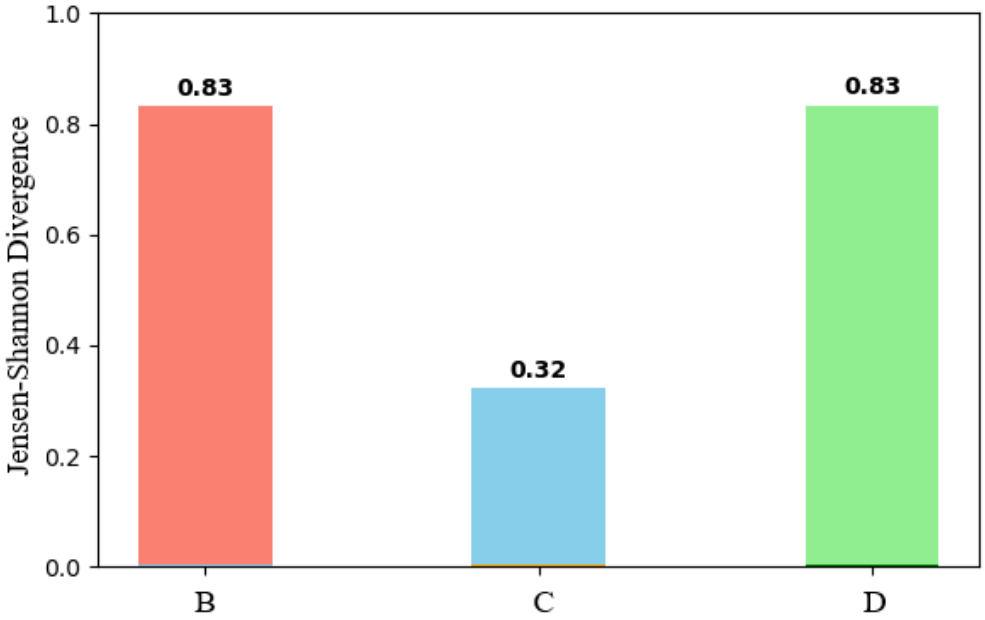

Figure 6.62: Datasets B, C, and D divergence value for prediction target value with respect to dataset A.

The target variable, like most features, is distributionally distant from dataset A in dataset B and D, whereas in dataset C it shows a much closer alignment.This might explain why, among the pairwise combinations using Strategy 2, A+C (in Strategy 2 learning curves subsection, 6.2.3) already shows better performance and a smoother learning curve, even without per-product scaling or binary labels in training, compared to A+B and A+D.

These differences between target variable distributions can be visualized in the figure (6.63) below:

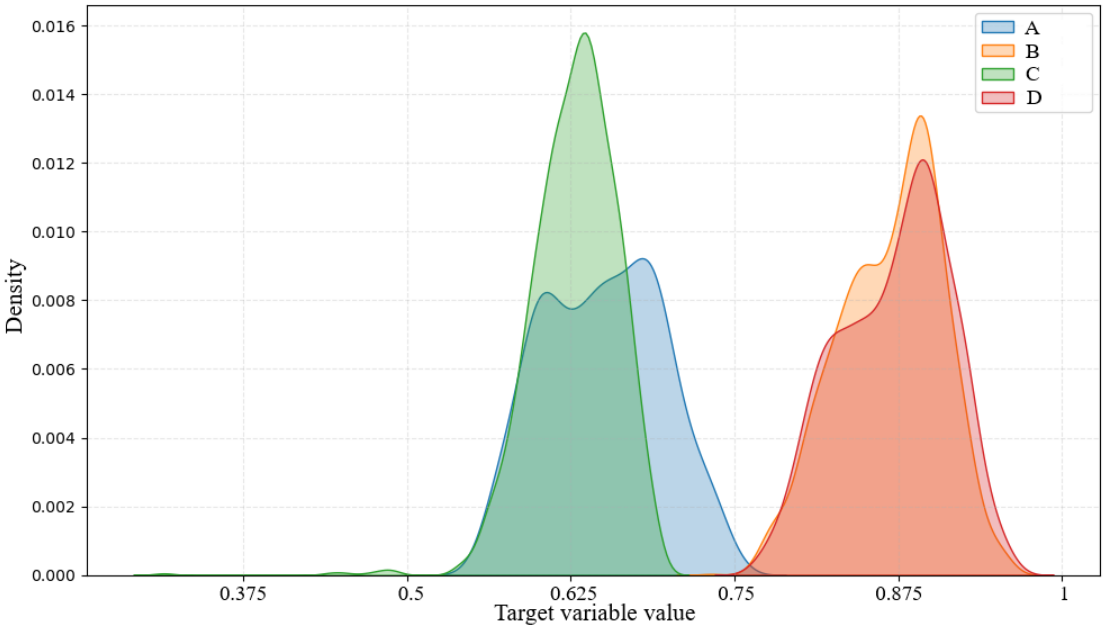

Figure 6.63: Prediction target variable density plots for each dataset.

Key Takeaways

Despite noticeable differences in feature and target distributions across dataset, combining datasets in model training, as shown in section 6.2 and especially in the key takeaways of incremental strategies section (6.2.3), still improves performance. By adopting careful strategies for sample weighting, scaling and feature selection/preprocessing the models are still able to extract useful patterns even when distributions diverge, without precluding effective multi-dataset learning. This basically means that by training on multiple datasets together, the model is exposed to a broader variety of input distribution, which, with a carefully selected configuration, enhances robustness as additional target-specific samples are incorporated, as it happens for the incremental training strategies described in this work.

6.3 Active Learning Integration

The optimal setups determined under Strategy 2 were replicated, with the integration of Active Learning as the sampling method for incremental training. The same experiments were naturally performed on other dataset combinations as well but did not produce relevant results. In this section, the main comparison focuses on the differences between the best configurations with and without Active Learning.

- **A+B+C (S2 Ridge Model): With vs Without Active Learning**

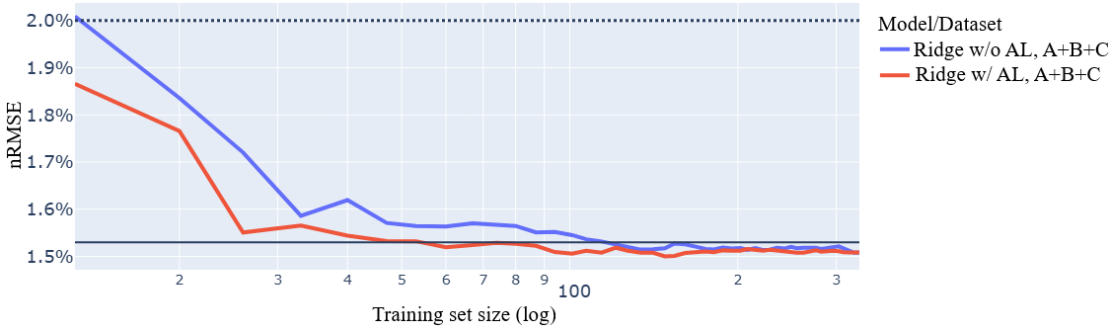

Figure 6.64: Comparison of nRMSE from S2 Ridge A+B+C with and without AL.

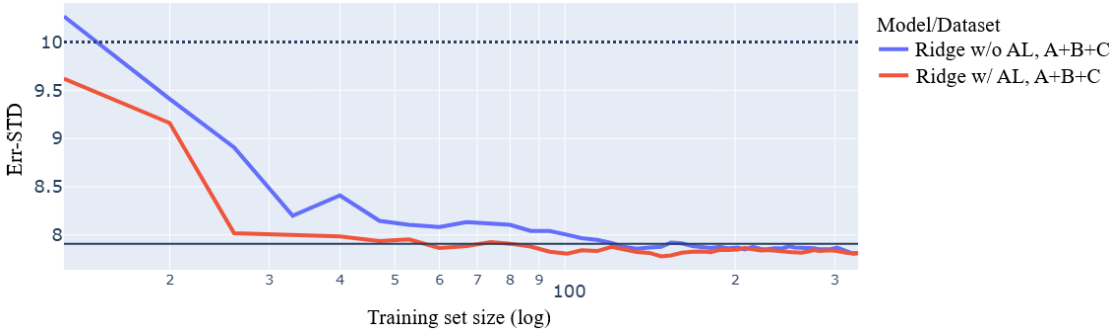

Figure 6.65: Comparison of err-STD from S2 Ridge A+B+C with and without AL.

In this case, integrating Active Learning brought visible benefits to the model. In the table below, the metrics from the two curves, at specific training set sizes, to allow a direct numerical comparison:

Model / AL	Metric	13	26	33	98	330
Ridge A+B+C w/ AL	nRMSE	1.87%	1.55%	1.57%	1.51%	1.51%
	err-STD	9.62	8.02	8.00	7.81	7.81
Ridge A+B+C w/o AL	nRMSE	2.01%	1.72%	1.59%	1.55%	1.51%
	err-STD	10.27	8.91	8.20	8.00	7.81

Table 6.16: Comparison of nRMSE and err-STD at selected training set sizes for Ridge models (Strategy 2) on the A+B+C dataset, with and without active learning (AL).

These results (where 330 corresponds to the maximum available samples from dataset A in training) confirm that, in the case of Ridge A+B+C with incremental Strategy 2, integrating Active Learning further improved accuracy especially in the lower range of used samples in training. Here is a comparison with the baseline reference learning curve (Ridge, A), where it is even more evident how the Active Learning version achieves roughly the same accuracy of the baseline reference but with a much smaller training set size:

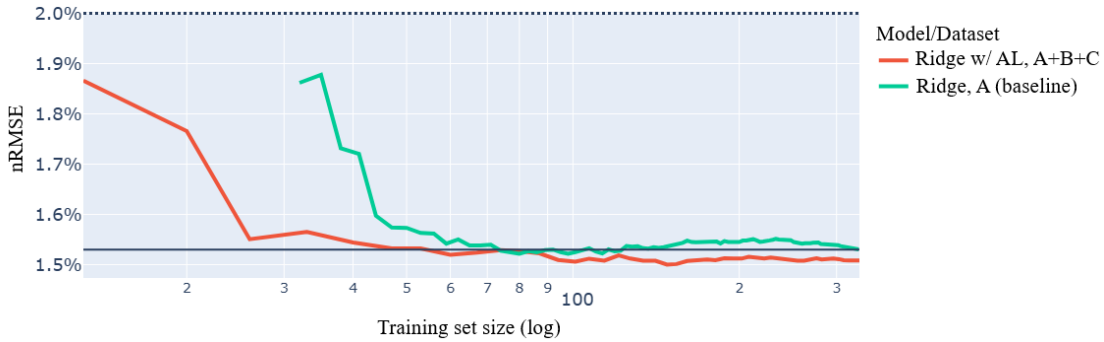

Figure 6.66: Comparison of nRMSE: S2 Ridge A+B+C with AL vs baseline model (Ridge, A).

- **A+B+C+D (S2 Ridge Model): With vs Without Active Learning**

Figure 6.67: Comparison of nRMSE from S2 Ridge A+B+C+D with and without AL.

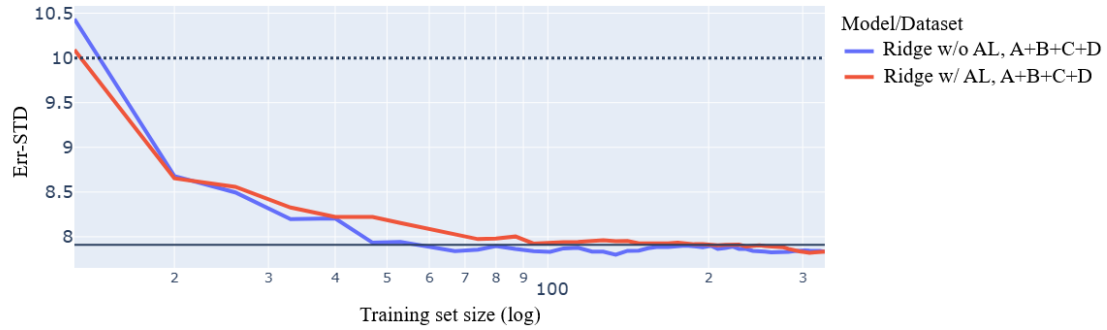

Figure 6.68: Comparison of err-STD from S2 Ridge A+B+C+D with and without AL.

In this case, Active Learning seems to have underperformed compared to its counterpart. While the learning curve indicates that the model still performs well with smaller training set sizes, it no longer stands out as the best. Between 45 and 230 A samples in training, its accuracy consistently falls slightly below the baseline reference, as illustrated below:

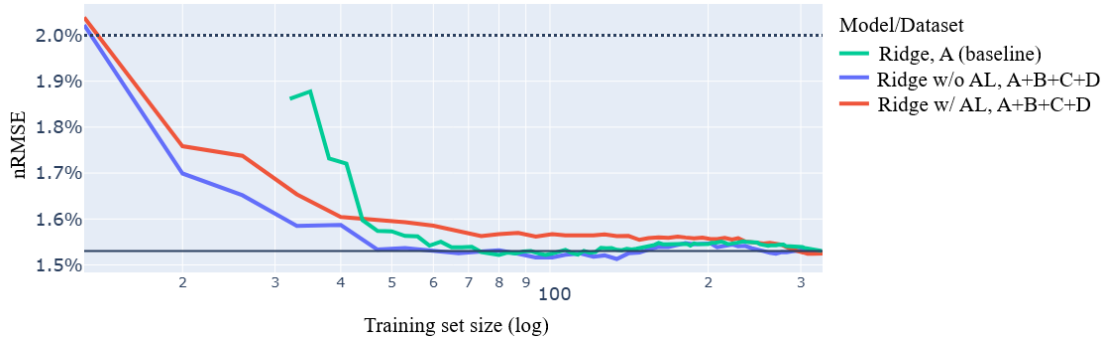

Figure 6.69: Comparison of nRMSE from S2 Ridge A+B+C+D with and without AL and with baseline reference.

A plausible explanation for why integrating AL in this configuration actually hurt performance might lie in the underlying distribution mismatch among datasets. Given the analysis performed on dataset similarity (Subsection 6.2.4), in the case of A+B+C, where the baseline training set is composed of B and C, the combination offers a reasonable compromise since dataset B introduces variability, while dataset C shows a target distribution that partially aligns with A. This balance allows Active Learning to effectively identify and select informative samples from A, ultimately improving the model's predictive performance. Conversely, when the baseline training set includes also dataset D (A+B+C+D case), the overall distributional distance becomes too large, introducing excessive heterogeneity. In this scenario, the AL mechanism probably struggles to select good informative samples for the target A, often selecting extreme or uninformative outliers instead, which ultimately negatively affect the training process and performance.

- **A+C (S2 Ridge Model): With vs Without Active Learning**
  Given the results obtained with this combination (Subsection 6.2.3 figs. 6.42 and 6.44) both with and without per-product scaling and binary labels, it was worth analyzing the effect of Active Learning also in this case.

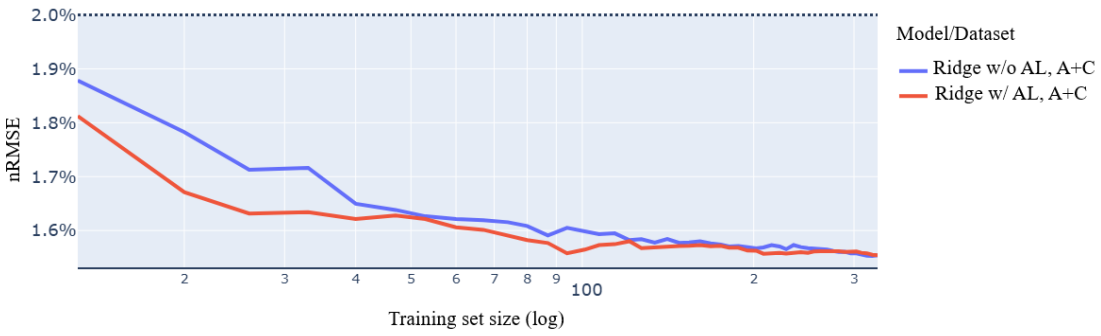

Figure 6.70: Comparison of nRMSE from S2 Ridge A+C with and without AL.

Figure 6.71: Comparison of err-STD from S2 Ridge A+C with and without AL.

Similar to A+B+C, this combination demonstrates measurable benefits from the integration of Active Learning. These improvements are quantitatively detailed in the table below.

Model / AL	Metric	13	26	33	98	330
Ridge A+C w/ AL	nRMSE	1.81%	1.63%	1.63%	1.56%	1.55%
	err-STD	9.12	8.37	8.30	8.07	8.03
Ridge A+C w/o AL	nRMSE	1.88%	1.71%	1.72%	1.60%	1.55%
	err-STD	9.56	8.76	8.73	8.21	8.03

Table 6.17: Comparison of nRMSE and err-STD at selected training set sizes for Ridge models (Strategy 2) on the A+C dataset, with and without active learning (AL).

Figure 6.72: Comparison of nRMSE from S2 Ridge A+C with and without AL and with baseline reference.

As shown in figure 6.72, this combination together with Active Learning integration achieves good accuracy, reaching the baseline reference performance with a considerably smaller number of samples. This advantage persists up to approximately 45 samples, beyond which the performance for both non-AL and AL versions slightly declines, remaining consistently below the baseline until eventually plateauing around comparable error levels.

6.3.1 Comparison of the Best Performing Configurations

To determine the overall best-performing configuration, the three most promising setups are compared in this section: A+B+C with Active Learning, A+B+C+D without Active Learning, and A+C with Active Learning.

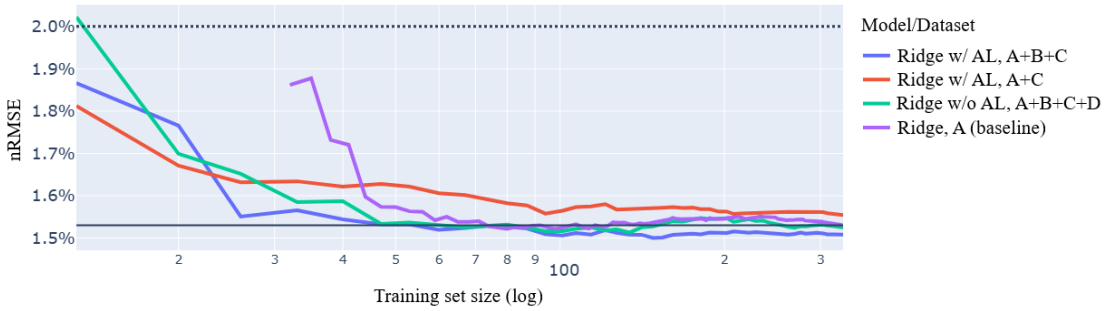

Figure 6.73: Comparison of nRMSE from the three best performing configurations and baseline reference.

From the figure above, it is immediately evident that the A+B+C with Active Learning configuration achieves the lowest overall error. However, when focusing on smaller training set sizes, the A+C with Active Learning curve actually yields the lowest error in the early stages. To better highlight these differences, the analysis below concentrates on the lower range of training samples (approximately 50 and below).

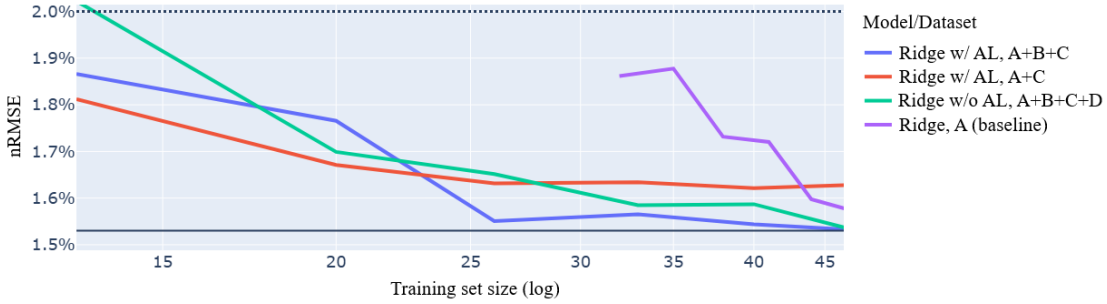

Figure 6.74: Comparison of nRMSE from the three best performing configurations and baseline reference (lower range, 50 samples and below).

Considering all aspects, the configuration that achieves the highest accuracy with the smallest number of training samples is A+C with Active Learning. However, when evaluating overall performance, the configuration A+B+C with Active Learning stands out as the most effective setup. It consistently exhibits the lowest

error values across the full training range, remaining well below both the baseline and the other configurations. In contrast, the A+B+C+D configuration performs very close to the baseline after around 70 samples, while A+C, despite its early advantage at low training set sizes, gradually converges and remains consistently above the baseline after around 45 training samples.

Therefore, **A+B+C with Active Learning (Strategy 2)** can be identified as the best overall configuration, offering the most stable and accurate results.

Model / Dataset	Training size	R^2	MAPE	nRMSE	nMAE	Err-STD
Baseline Ridge (A)	13	-	-	-	-	-
	26	-	-	-	-	-
	33	0.92	1.47%	1.86%	1.48%	9.60
	100	0.95	1.20%	1.52%	1.20%	7.85
	330	0.95	1.20%	1.53%	1.20%	7.91
Ridge AL A+B+C	13	0.93	1.49%	1.87%	1.49%	9.62
	26	0.95	1.24%	1.55%	1.25%	8.02
	33	0.95	1.26%	1.57%	1.26%	8.00
	100	0.95	1.21%	1.51%	1.21%	7.81
	330	0.95	1.22%	1.51%	1.22%	7.81

Table 6.18: Comparison at selected training set sizes for Ridge models (Strategy 2) on the A+B+C dataset with AL and baseline reference. Metrics shown: R^2, MAPE, nRMSE, nMAE, Err-STD.

6.4 Complementary Experiments

These additional experiments were performed using only the best configuration identified in the previous analyses (A+B+C with Active Learning), as extending them to all configuration would not have been meaningful. Their purpose was to investigate whether the optimal setup could be further improved. In particular, alternative scaling strategies were tested to assess potential benefits in predictive accuracy, while the CORAL (Correlation Alignment) method was employed as a domain adaptation technique to evaluate its effect on model performance. Lastly, an additional analysis was carried out to explore the model's behavior with respect to wafer types through a dedicated traning process.

6.4.1 Alternative Scalers Evaluation

These experiments were conducted using the same pipeline and configuration that produced the optimal result (A+B+C with Active Learning). However, the scaling method within the per-product strategy was varied, replacing the `StandardScaler` with the alternative techniques listed below.

- `MinMaxScaler`

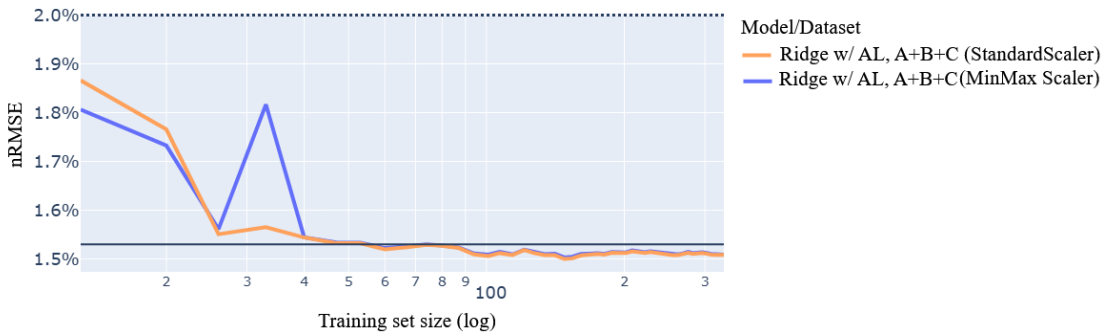

Figure 6.75: nRMSE learning curve comparison for the optimal configuration (A+B+C with Active Learning, Strategy 2), highlighting the effect of using `StandardScaler` versus `MinMaxScaler` within the per-product scaling strategy.

Applying MinMaxScaler led to slightly better performance for very small training set sizes (below 20 sample) but introduced an evident error spike between 25–40 samples, after which it converged to the same error plateau as the optimal configuration.

- `RobustScaler`

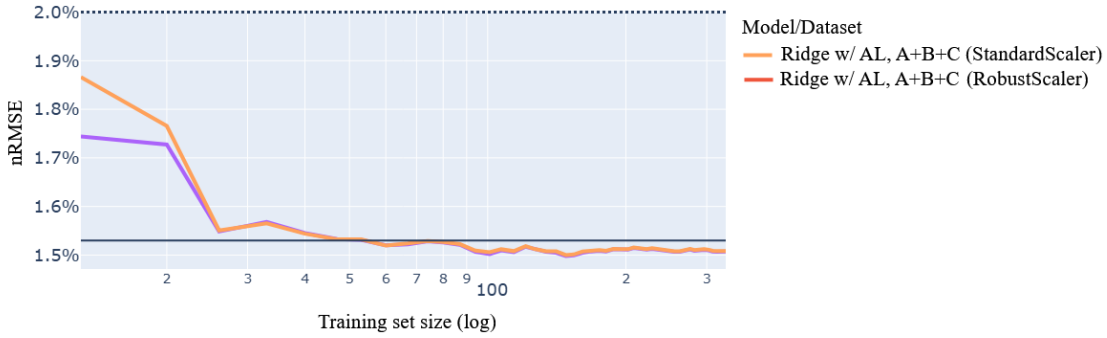

Figure 6.76: nRMSE learning curve comparison for the optimal configuration (A+B+C with Active Learning, Strategy 2), highlighting the effect of using `StandardScaler` versus `RobustScaler` within the per-product scaling strategy.

The RobustScaler produced an almost identical learning curve to the optimal one, but with a lower initial error, suggesting improved stability on small training set sizes due to its robustness to outliers.

- `MaxAbsScaler`

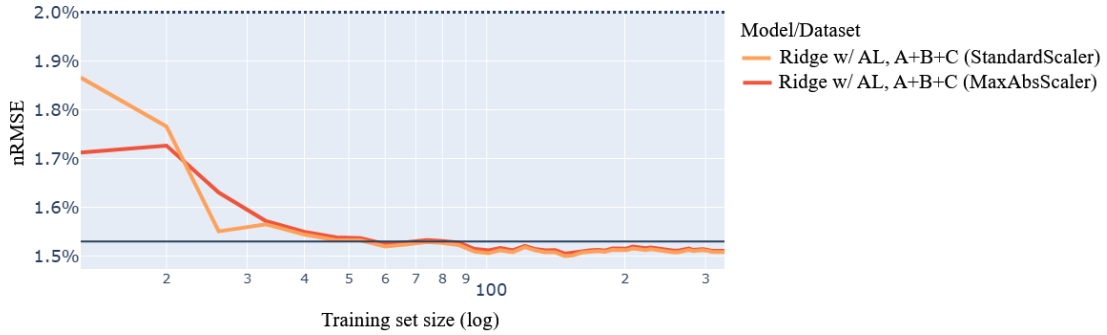

Figure 6.77: nRMSE learning curve comparison for the optimal configuration (A+B+C with Active Learning, Strategy 2), highlighting the effect of using `StandardScaler` versus `MaxAbsScaler` within the per-product scaling strategy.

The MaxAbsScaler behaves similarly to the RobustScaler but with slightly higher errors in the 20-35 sample range, remaining nonetheless a reasonable alternative.

Final Considerations on Alternative Scalers

Among the scalers tested, **RobustScaler** provides the best performance, slightly reducing the initial error in lower ranges of training set size while mantaining the same overall learning curve as the optimal StandardScaler configuration. MinMaxScaler and MaxAbsScaler also perform reasonably well, with minor deviations in small training sizes. Overall, applying especially RobustScaler in the per-product scaling strategy allows for a further refinement of the already high-performing model (A+B+C with Active Learning, Strategy 2) enhancing stability and reducing initial error without altering the plateau performance. When using the `RobustScaler`, the improvement over the second-best optimal model becomes even more pronounced, as illustrated in the figure below (6.78).

Scaler	Training size	R^2	MAPE	nRMSE	nMAE	Err-STD
StandardScaler	13	0.93	1.49%	1.87%	1.49%	9.62
	26	0.95	1.24%	1.55%	1.25%	8.02
	33	0.95	1.26%	1.57%	1.26%	8.00
	100	0.95	1.21%	1.51%	1.21%	7.81
	330	0.95	1.22%	1.51%	1.22%	7.81
RobustScaler	13	0.94	1.39%	1.74%	1.40%	9.01
	26	0.95	1.24%	1.55%	1.25%	8.01
	33	0.95	1.27%	1.57%	1.27%	8.00
	100	0.95	1.20%	1.50%	1.20%	7.79
	330	0.95	1.22%	1.51%	1.22%	7.81

Table 6.19: Comparison at selected training set sizes for the optimal configuration (A+B+C w/ AL, Strategy 2) using StandardScaler versus RobustScaler in the per-product scaling strategy. Metrics shown: R^2, MAPE, nRMSE, nMAE, and Err-STD.

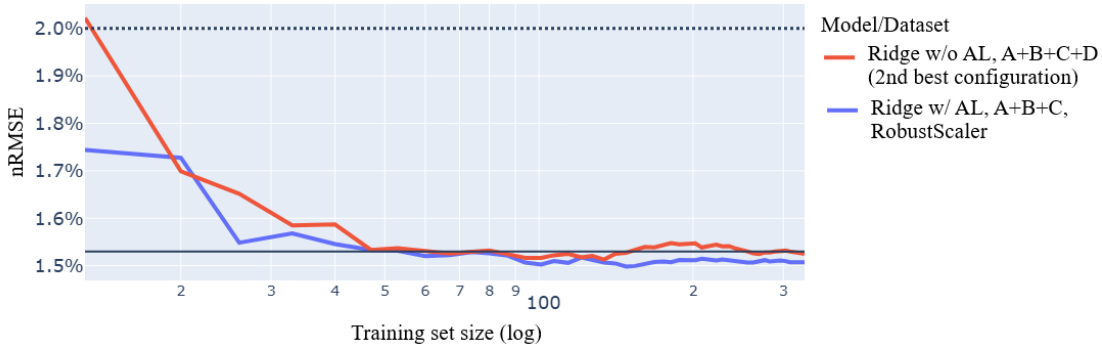

Figure 6.78: nRMSE learning curve comparison for the optimal configuration (A+B+C with Active Learning, Strategy 2), highlighting the improvement of using `RobustScaler` within the per-product scaling strategy versus second best optimal configuration (A+B+C+D, without Active Learning, Strategy 2).

6.4.2 CORAL Experiments

The learning curves experiments were conducted using the same pipeline as before, with per-product scaling and Active Learning. CORAL alignment was applied after scaling to achieve correlation alignment of the samples from older generations and the A samples (increasing at each iteration).

No additional plots are included here, as the learning curves for this configuration is essentially overlapping with the one obtained from the optimal configuration. This indicated that the combination of per-product scaling with labels in training (through a custom scaler applying RobustScaler) does not introduce any significant changes.

6.4.3 Wafer-type Based Training

To further investigate how the model behaves with respect to different wafer production stages, additional experiments were performed considering the wafer type as criteria to distinguish samples. In semiconductor manufacturing, wafers are typically categorized into production wafers (fully processed lots used for standard production), and split-lot wafers (from partially processed lots used mainly for testing and characterization purposes). These two different categories represent distinct points in the manufacturing flow.

The purpose of these experiment was to analyze how model performance evolves as samples from these two wafer types are incrementally introduced in the training set. Specifically, following an incremental approach with Active Learning, production samples were initially added into the training set, and once all of them were included, the split-lot samples were added to training set in the same manner. For the multi-dataset scenario, the Strategy 2 incremental approach was applied (older-generation datasets as baseline training set expanded by incrementally adding samples from dataset A). This approach allows assessing how accuracy is influences by the nature and order of wafer data included in training.

The tests were carried out on three configurations:

1. the Ridge baseline model (trained only on dataset A).
2. the A+B+C+D Ridge model without Active Learning (second-best configuration under Strategy 2).
3. the A+B+C Ridge model with Active Learning using RobustScaler in the per-product scaling strategy (the configuration identified as the best one overall)

In the learning curves shown below, nRMSE for three different test sets can be seen: one for the mixed test set (both split-lot and production samples), one for a split-lot only samples test set, and one for a production only samples test set. The pool of production samples in all of three cases consists of 80 production samples, after which split-lot samples are incrementally added to the training set. It is, indeed, possible to notice major differences in plots starting exactly from the point where those split-lot samples are starting to be included.

Wafer-type Based Training: Ridge Baseline Model

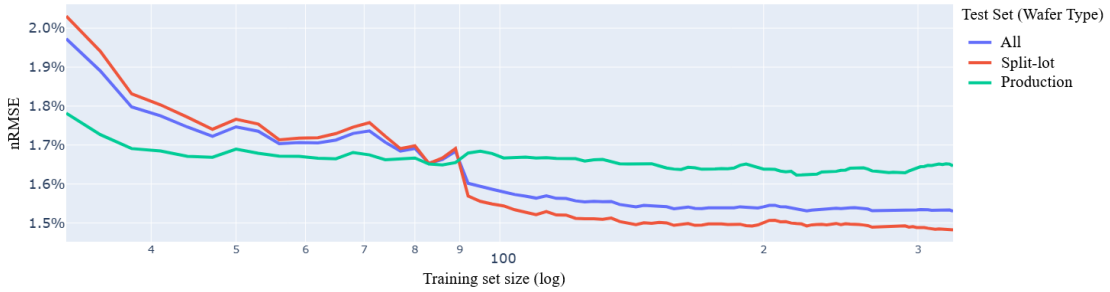

Figure 6.79: Comparison of nRMSE plots (learning curves) of baseline reference model (Ridge, A) for mixed, split-lot only, and production only test sets.

Wafer-type Based Training: Ridge without AL, A+B+C+D

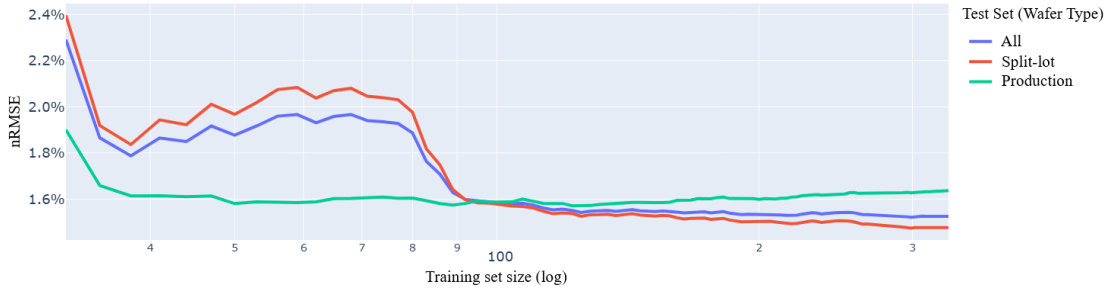

Figure 6.80: Comparison of nRMSE plots (learning curves) of second best configuration (Ridge, A+B+C+D, without AL, Strategy 2) for mixed,split-lot only, and production only test sets.

Wafer-type Based Training: Ridge with AL, A+B+C

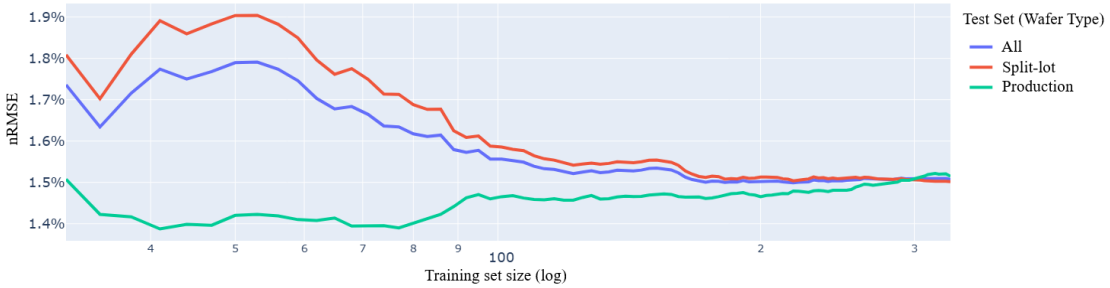

Figure 6.81: Comparison of nRMSE plots (learning curves) of second best configuration (Ridge, A+B+C, without AL, Strategy 2, RobustScaler) for mixed,split-lot only, and production only test sets.

Key Takeaways for Wafer-type Based Training

For this experiments, the main focus is placed on the learning curve obtained by evaluating the model on production wafer samples. This choice reflects the nature of the manufacturing process itself, where in the early stages, the majority of available data originates from production wafers, while split-lot wafers become available only later.

The baseline model already achieves strong performance, with an initial nRMSE of approximately 1.8% and a final value around 1.65%. However, the multi-dataset models consistently outperform the baseline, even in this context. Among them, the already highlighted best configuration (Ridge with AL, A+B+C, Strategy2, RobustScaler in the per-product scaling strategy) delivers the best overall performance.

It is also important to note that when split-lot samples begin to be introduced into the training set, a slight decrease in accuracy on the production wafer test set can be observed. This effect is generally minor, so small that the learning curve remains nearly unchanged after few increments. However, this trend does not hold for the best-performing configuration, where instead the accuracy on the production test set deteriorates more noticeably after adding split-lot samples. In this case, the curve gradually plateaus, converging with the curves obtained from the mixed and split-lot only test sets, all reaching a similar nRMSE of approximately 1.5%.

Additional evaluation metrics summarizing these observations are reported in the Table 6.20 below. The best configuration (Ridge, A+B+C with AL and RobustScaler, Strategy 2) has been called "R-AL-R,A+B+C" , while the second best configuration (Ridge, A+B+C+D without AL, Strategy 2) has been named "R-nAL,A+B+C+D" (these short labels have been adopted for brevity within the table).

Model / Dataset	Training size	#P	#S	R^2	nRMSE	Err-STD
Baseline Ridge (A)	32	32	0	0.72	1.78%	8.98
	65	65	0	0.75	1.66%	8.62
	98	79	19	0.75	1.68%	8.75
	170	79	91	0.76	1.64%	8.67
	330	79	247	0.75	1.65%	8.71
R-nAL, A+B+C+D	32	32	0	0.62	1.90%	10.15
	65	65	0	1.76	1.60%	8.51
	98	80	18	0.77	1.59%	8.38
	170	80	90	0.77	1.60%	8.34
	330	80	250	0.76	1.64%	8.23
R-AL-R, A+B+C	32	32	0	0.79	1.51%	8.22
	65	65	0	0.81	1.41%	7.65
	98	80	18	0.80	1.46%	7.87
	170	80	90	0.80	1.46%	7.88
	330	80	250	0.78	1.51%	8.08

Table 6.20: Comparison at selected training set sizes of best configurations and and baseline reference (production wafer-only test set). Metrics shown: R^2, nRMSE, Err-STD. "#P" and "#S" indicate the number of production and split-lot samples in the training set.

Chapter 7

Conclusion

This work presented an empirical exploration of incremental and active learning strategies for multi-dataset regression applied to MCU performance prediction, investigating how different preprocessing, scaling, and sampling strategies impact model behavior and accuracy. The experiments progressively built on each other, revealing key insights into dynamics of scaling effects, dataset merging, and active sample selection.

7.1 Summary of Findings

Initial experiments on the single-dataset baseline (Ridge, dataset A) confirmed that even simple linear models, when combined with appropriate preprocessing, can achieve high predictive performance, with $R^2 > 0.94$ and low error values (e.g. nRMSE, Err-STD ...). More complex models such as TabPFN did not provide significant advantages, validating the choice of lightweight models for incremental and multi-dataset scenarios, specifically in a hardware-constrained environment where computationally heavier models cannot be employed.

When expanding to multi-dataset configurations, results demonstrated that Strategy 2, where all older generation datasets form the baseline training set and new-generation samples are added incrementally, significantly outperforms Strategy 1, which relied on Active Learning from the start, but was based on a mixed training set of old-generation and new-generation samples all together. The addition of per-product scaling and binary product labels in training provided further improvements, producing smoother learning curves and lower overall error. Among all tested combinations, A+B+C+D emerged as the best configuration under Strategy 2, achieving the lowest error with relatively few samples and especially with less samples than the baseline reference.

Furthermore, the dataset similarity analysis clarified that, despite clear differences in both feature and target variable distributions among datasets, multi-dataset learning can still be highly effective, especially in situations in which available data is scarce. However, proper scaling, sample weighting, and preprocessing are essential to make this approach work and allow models to extract useful information and patterns from multiple datasets. The results obtained overall confirmed that distributional differences does not preclude effective learning. On the contrary, diversity in data might actually be enhancing robustness and generalization, as long as the relationship between features and prediction target are similar across datasets and at least a fraction of the data is distributionally close.

Experiments on Active Learning (AL) integration in Strategy 2 produced improved models, although performance gains were not uniform across all configurations. These tests showed that A+C (with AL) combination achieved the highest accuracy with the smallest number of samples (probably also because dataset C was the most similar to dataset A), while A+B+C (with AL) provided the best overall and most stable performance across all training ranges, maintaining the lowest errors visibly below the baseline. Therefore, this configuration represents the most effective overall setup, combining the benefits of multi-dataset learning with the efficiency of an active sample selection approach.

The experiments performed with alternative scaling approaches revealed that the RobustScaler, within the per-product scaling strategy, resulted in improved performance, slightly reducing early-stage errors while maintaining the same behavior and plateau accuracy as StandardScaler. This result highlights how the scaling choice in this context can influence model performance.

Tests with CORAL (Correlation Alignment), aimed at domain adaptation between source and target datasets, showed no benefits in this context. When applied, infact, in combination with per-product scaling, CORAL produced basically identical results to the version without CORAL. This suggests that the existing per-product scaling already achieves a substantial degree of domain normalization and that methods such as CORAL might offer low additional benefits when datasets are already properly handled through scaling strategies.

Finally, the wafer based training analysis provided additional insights into the model's behavior in relation to production stages. When trained on progressively added production and split-lot wafers samples, the baseline Ridge model performed well, but the multi-dataset models again achieved slightly higher accuracy. From the learning curves it was clear that when split-lot samples were introduced, a slight, almost negligible, drop in accuracy on production wafers was observed. Also in this case, the best configuration was the same found in previous experiments: Ridge with AL and RobustScaler within the per-product scaling approach with A+B+C datasets, under Strategy 2, where the overall nRMSE remained low (1.5%), confirming the robustness of the best model even with mixed wafers.

7.2 Final Considerations

This work achieved and demonstrated tangible improvements with multi-dataset and active learning strategies, showing their effectiveness. Several open challenges and opportunities to extend the analysis in this context still remain. One direction to further analyze lies in domain adaptation, where more advanced nonlinear alignment methods, such as deep CORAL, could better handle complex inter-dataset shifts and improve cross-generation generalization. Similarly, also the Active Learning strategy based on Hausdorff distance could be enhanced by integrating selection based on uncertainty or hybrid criteria to prevent the model from prioritizing outliers in a context of heterogeneous data.

From a methodological point of view, future work could explore richer model architectures, such as neural networks, to assess whether their increased capability of finding pattern and information can further leverage the data across different datasets.

Beyond these potential refinements, the significance of this research lies in demonstrating how Machine Learning approaches can directly support the optimization of semiconductor manufacturing. By integrating an incremental learning approach, Active Learning, proper scaling, and multi-dataset fusion to leverage prior knowledge, this work shows how these algorithms can be effectively embedded into industrial workflows to predict device performance, improving production efficiency and making product characterization easier, especially given the predictive performance and convergence achieved with reduced data requirements.

Furthermore, while the findings of this work are focused on the specific context of MCU production, they extend more broadly, demonstrating how carefully designed learning pipelines and domain-aware modeling can produce algorithms that serve as core analytical components supporting adaptive and autonomous screening for efficient industrial production.

Bibliography

[1] Nicolò Bellarmino. «Computational Intelligence Techniques for Device Testing». MA thesis. Politecnico di Torino, 2021 (cit. on pp. 3, 21, 25, 29, 30).

[2] Riccardo Cantoro, Martin Huch, Tobias Kilian, Raffaele Martone, Ulf Schlichtmann, and Giovanni Squillero. «Machine Learning based Performance Prediction of Microcontrollers using Speed Monitors». In: *2020 IEEE International Test Conference (ITC)*. 2020, pp. 1–5. DOI: 10.1109/ITC44778.2020.9325253 (cit. on p. 3).

[3] Nicolò Bellarmino, Riccardo Cantoro, Martin Huch, Tobias Kilian, Raffaele Martone, Ulf Schlichtmann, and Giovanni Squillero. «Exploiting Active Learning for Microcontroller Performance Prediction». In: *2021 IEEE European Test Symposium (ETS)*. 2021, pp. 1–4. DOI: 10.1109/ETS50041.2021.9465472 (cit. on p. 3).

[4] Nicolò Bellarmino, Riccardo Cantoro, Martin Huch, Tobias Kilian, Ulf Schlichtmann, and Giovanni Squillero. «Feature Selection for Cost Reduction in MCU Performance Screening». In: *2023 24th IEEE Latin-American Test Symposium (LATS)*. Veracruz, Mexico, Mar. 2023, pp. 1–6. DOI: 10.1109/LATS58125.2023.10154495 (cit. on p. 6).

[5] Nicolò Bellarmino, Riccardo Cantoro, Martin Huch, Tobias Kilian, and Giovanni Squillero. «Grouped Feature Selection for SMONs Placement in MCU Performance Screening». In: *2025 IEEE 26th Latin American Test Symposium (LATS)*. San Andrés, Colombia, Mar. 2025, pp. 1–6. DOI: 10.1109/lats65346.2025.10963942 (cit. on p. 6).

[6] 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 (cit. on p. 11).

[7] Duanyang Liu, Liming Xu, Xumin Lin, Xing Wei, Wenjie Yu, Yang Wang, and Zhongming Wei. «Machine Learning for Semiconductors». In: *Chip* 1 (Nov. 2022), p. 100033. DOI: 10.1016/j.chip.2022.100033 (cit. on p. 11).

[8] 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 (cit. on p. 11).

[9] Chen He, Hanbin Hu, and Peng Li. «Applications for Machine Learning in Semiconductor Manufacturing and Test (Invited Paper)». In: Apr. 2021, pp. 1–3. DOI: 10.1109/EDTM50988.2021.9420935 (cit. on p. 11).

[10] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 1st. New York: Springer, 2006, pp. 137–143 (cit. on pp. 13, 15).

[11] Fabian Pedregosa et al. «Scikit-learn: Machine Learning in Python». In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 13).

[12] Scikit-learn developers. *Scikit-learn: User Guide – Preprocessing data*. https://scikit-learn.org/stable/modules/preprocessing.html. Accessed: 2025-09-20 (cit. on p. 13).

[13] I. T. Jolliffe. *Principal Component Analysis*. 2nd. Springer, 2002 (cit. on p. 14).

[14] Scikit-learn developers. *Polynomial Features — Scikit-learn 1.3.2 documentation*. Accessed: 2025-09-20. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html (cit. on p. 15).

[15] Burr Settles. «Active Learning Literature Survey». In: *University of Wisconsin, Madison* 52 (July 2010) (cit. on p. 18).

[16] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil Lawrence. «Dataset Shift in Machine Learning». In: (Jan. 2009) (cit. on p. 18).

[17] Baochen Sun, Jiashi Feng, and Kate Saenko. «Return of Frustratingly Easy Domain Adaptation». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2016 (cit. on p. 19).

[18] Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and W. Kegelmeyer. «SMOTE: Synthetic Minority Over-sampling Technique». In: *J. Artif. Intell. Res. (JAIR)* 16 (June 2002), pp. 321–357. DOI: 10.1613/jair.953 (cit. on p. 19).

[19] Sinno Jialin Pan and Qiang Yang. «A Survey on Transfer Learning». In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191 (cit. on p. 19).

[20] Jesse Read, Bernhard Pfahringer, Geoffrey Holmes, and Eibe Frank. «Classifier Chains for Multi-label Classification». In: vol. 85. Aug. 2009, pp. 254–269. ISBN: 978-3-642-04173-0. DOI: 10.1007/978-3-642-04174-7_17 (cit. on p. 23).

[21] Scikit-learn developers. *LeavePGroupsOut*. Accessed: 2025-09-21. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeavePGroupsOut.html (cit. on p. 24).

[22] Scikit-learn developers. *StandardScaler*. Accessed: 2025-09-22. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (cit. on p. 29).

[23] Scikit-learn developers. *MinMaxScaler*. Accessed: 2025-09-22. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (cit. on p. 29).

[24] Scikit-learn developers. *RobustScaler*. Accessed: 2025-09-22. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (cit. on p. 29).

[25] Scikit-learn developers. *QuantileScaler*. Accessed: 2025-09-22. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (cit. on p. 29).

[26] Scikit-learn developers. *MaxAbs*. Accessed: 2025-09-22. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (cit. on p. 29).

[27] Scikit-learn developers. *PCA*. Accessed: 2025-09-22. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (cit. on p. 30).

[28] Scikit-learn developers. *TrainTestSplit*. Accessed: 2025-09-22. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (cit. on p. 32).