Deep Learning Strategies for Labeling and Accuracy Optimization in Microcontroller Performance Screening

Deep Learning Strategies for Labeling and Accuracy Optimization in Microcontroller Performance Screening

Nicolò Bellarmino*, Riccardo Cantoro*, Martin Huch†,
Tobias Kilian†‡, Ulf Schlichtmann‡ and Giovanni Squillero*

***Abstract*—In safety-critical applications, microcontrollers must be compliant with the required quality constraints and performance standards, particularly in terms of the maximum operating frequency ($F_{\max}$). Machine learning models have proven effective in estimating $F_{\max}$ by utilizing data extracted from on-chip ring oscillators (ROs), making them a valuable instrument for performance screening. However, the cost of obtaining labeled samples and the stringent accuracy needed by the model create hard challenges in this context. In order to address these, we explored three deep-learning-based key strategies:**

- **Semi-Supervised Learning with Deep Feature Extractors: we leverage the abundance of unlabeled production data in a semi-supervised approach. Deep feature extractor models are employed to transform data into higher-dimensional spaces. These feature embeddings enable accurate performance prediction using simple linear regression, with a fraction of labeled data to reach baseline performances.**
- **Intra-family Transfer Learning: when introducing new microcontroller products, with slightly different characteristics but the same set of ROs, previously trained deep feature extractors can be used, in a transfer learning fashion. This permits the use of significantly fewer labeled data compared to traditional methods.**
- **Inter-family Transfer Learning: we extend the previous transfer learning concept to new microcontroller products with completely distinct characteristics. We aim to demonstrate that adapting the features set and fine-tuning deep learning feature extractors initially trained on specific legacy product data permits to yield better performance.**

Our research aims to provide a holistic framework for deep-learning-based microcontroller performance screening to address the challenge of limited labeled data. The proposed methodologies significantly improve prediction accuracy and reduce the dependency on a large number of labeled samples, thus enhancing the efficiency and efficacy of machine-learning-based microcontroller screening. The proposed framework enables models re-use, serving as a valuable baseline when new products are released.

***Index Terms*—Fmax, Speed Monitors, Ring Oscillators, Speed Binning, Machine Learning, Device Testing, Manufacturing, Transfer Learning, Deep Learning**

I. Introduction

In safety-critical industries like automotive and aerospace, the reliable performance of microcontrollers (MCUs) is paramount. Traditionally, the screening of MCUs has revolved around identifying devices that meet specific criteria, particularly the maximum operating frequency ($F_{\max}$). Conventional methods involve extensive testing at increasing clock frequencies, a process that is not only time-consuming but also relies on expensive test setups. Moreover, it only provides categorical results. In response to these challenges, machine learning (ML) regression models have been proposed to predict $F_{\max}$ of MCUs based on correlated data, offering significant time savings compared to traditional methods [1]: oscillation frequencies values from Ring Oscillators (ROs) can be used to predict $F_{\max}$ in different test cases (*labels*) by using ML models. This latter technique is not intended to replace structural testing, yet it is a much more informative process compared to a simple speed binning [2]–[7]. A crucial factor in the accuracy of supervised ML models is the availability of high-quality labeled data. Unfortunately, collecting $F_{\max}$ data is a laborious task, and it can be challenging. Constructing a new ML model from scratch under such circumstances is time-consuming and not always feasible. Moreover, using a pre-existing model trained on previous-generation data may not yield accurate results due to shifts in data distributions, introducing the need to re-train or fine-tune the model. This paper addresses these challenges by optimizing the ML model training process, reducing the need for a large number of labeled samples, minimizing prediction errors, and defining a framework to build predictive models that can be re-used among product families. To achieve this, we employed deep learning (DL) to extract relevant information and patterns from unlabeled data collected from production lines, using semi-supervised learning (SSL) and Transfer Learning approaches. DL models are used as feature transformers. Resulting data extracted by DL models are then utilized to train a simple Linear Regression model to predict $F_{\max}$. The same deep models, originally developed for one MCU product (*A1*), are also leveraged to create predictive models for a different product (*A2*), in a Transfer Learning fashion. Furthermore, as an extension of previous work on Inter-Family Transfer Learning [8], [9], we explored Transfer Learning techniques from a legacy product family (*A*) to address data scarcity when releasing a new MCU product (*B1*) of a different family *B*, effectively reducing the effort and time required for label

* Politecnico di Torino (Turin, Italy). † Infineon Technologies AG (Munich, Germany). ‡ Technical University of Munich (Munich, Germany). Authors are listed in alphabetical order.

acquisition. The experiments conducted in this study demonstrate that these approaches substantially reduce the number of labeled samples needed to build accurate models and enhance their performance compared to traditional shallow-learning baseline. This accelerates the deployment of ML predictive models in production. Leveraging insights and models from previous products saves substantial time in the ongoing research and development process. Our proposed framework establishes a solid groundwork for further investigation. With each new product, testing and machine learning engineers can explore novel models and techniques to refine performance prediction accuracy. In the absence of comprehensive insights regarding data, SMONs interaction, and the necessary number of labeled samples for training reliable models, our framework serves as a dependable benchmark, representing a noteworthy advancement in MCU performance screening. Our experiments involved real-world MCU products with the goal of the development of models that transcend individual product variations. By leveraging a diverse set of MCU samples, our framework aims to capture the inherent characteristics common to different products within the MCU domain. This approach not only enhances the generalizability of our findings but also addresses a gap in the existing literature. Code and models are available in a public GitHub repository [1].

The rest of the paper is organized as follows: Section II presents related works on the topic. Section III provides the necessary background information, including data collection processes for ML algorithms (Sections III-A to III-C) and an introduction to Deep Convolutional Networks, Semi-Supervised and Transfer Learning (Sections III-D to III-F). In Section IV, the motivations for deploying deep learning models are given. In Section V, we detail the proposed approach. Sections VI and VII present the experimental setup and evaluation. Finally, in Section VIII we draw the conclusions.

II. Related Work

In recent years, the utilization of Machine Learning in both design and testing has garnered considerable attention, with a multitude of data analytic methods based on ML being investigated [10], [11].

The interest in predicting the maximum operating frequency ($F_{\max}$) of MCUs in safety-critical applications has grown. The early utilization of ML models to establish a relationship between structural and functional $F_{\max}$, firstly introduced in [3], was studied by several researchers [1], [12], [13]. However, the majority of existing works rely on simulated data [5] or have analyzed a few samples (tens) of data from MCU [3].

In the literature, establishing a mapping between indirect measurements (that can be acquired with little effort) and circuit specifications is called "alternate test". In the past, this has been extensively analyzed in the analog domains, rather than for digital circuits. Notably, the mapping between indirect measurements, specifically on-chip ROs frequency values, and $F_{\max}$ has been a focal point of study in MCU performance screening [1]. Appropriate ROs oscillation frequencies can be measured via software during the productive test flow. ROs frequencies can be used as Speed MONitors (SMONs), having the potential to capture variations in the physical parameters, enabling the prediction of the performance of the MCUs [1], [14].

Researchers have further explored ML models for $F_{\max}$ prediction in the MCU domain. In [1], they correlated 27 SMONs values from wafer sorting with functional $F_{\max}$. In [14], they found that polynomial ridge regression (Poly Ridge) effectively serves as an ML model for MCU performance screening. Active Learning (AL) was employed in [15] to reduce the training set size by selecting informative samples for model derivation, and outlier detection techniques were evaluated to identify noisy data and outliers [16]. The significance of feature selection in MCU performance screening was addressed in [17].

Moreover, the research community has delved into the realm of semi-supervised learning (SSL) and Transfer Learning [18]–[20], with deep neural networks playing a pivotal role in contemporary ML research. Various Deep SSL methods have been developed to leverage unlabeled data effectively [21]. Transfer learning has gained prominence as a technique to apply knowledge from source domains to enhance performance on target tasks. The concept of pretraining and fine-tuning was introduced in [22], where a CNN is pre-trained on a large-scale dataset and then fine-tuned for a specific target task, yielding significant improvements in computer vision tasks. Meta-learning [23], [24] aims to rapidly adapt to new tasks or environments based on prior knowledge and limited training data. These concepts have potential applications in addressing the challenges posed by limited labeled data and data distribution shifts in MCU performance screening.

III. Background

A. MCU Performance Screening

Testing activities for integrated circuits (ICs) are conducted at various stages throughout their life cycle. Initial validation tests occur during the product design phase to ensure compliance with specifications. Characterization tests are then performed on the first prototypes to identify faults and potential failures, possibly leading to product redesign.

After passing characterization tests, large-volume production begins. On-wafer testing is done during production, before dicing and packaging. Post-packaging tests confirm the proper completion of this process, ensuring that the ICs meet quality standards for market release. Subsequent in-the-field tests are conducted to verify the correct functioning of specific applications.

The testing phase for ICs is resource-intensive, both in terms of time and money, with costs escalating rapidly as integration levels increase. Functional tests aim to verify adherence to specifications, but for digital microcontrollers, exhaustive testing of each component's functionality is often impractical [25].

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

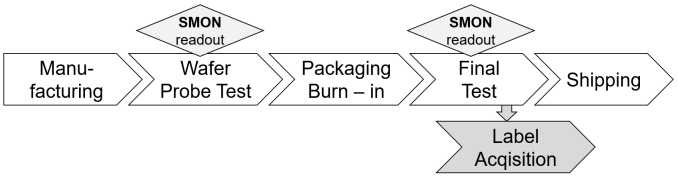

Fig. 1. Data collection steps through the manufacturing

For MCUs or System on Chip with at least one microprocessor, Software-Based Self-Test (SBST) [26] is a strategy wherein the processor executes suitable test programs to assess the device. These programs are loaded into the processor, stimulating specific circuit components, and collecting responses for evaluation. SBST is advantageous as it is non-intrusive, eliminates the need for extra hardware, and allows tests to be conducted at the processor's operating frequency. This approach enables defect screening at higher speeds, prevents overtesting, and is applicable throughout the product's lifetime in the field, without requiring expensive test equipment.

MCU Performance Screening involves evaluating and identifying devices based on their operational capabilities, specifically screening out those with the maximum operating frequency $F_{\max}$ below a predetermined threshold. This screening process ensures that only microcontrollers meeting or exceeding the specified frequency are considered suitable for further use or deployment. This method helps guarantee that selected devices can perform at the required speeds and meet the performance expectations outlined for a particular application or system.

B. Data Collection: Features

In the context of MCU performance screening, the ML training process involves acquiring a suitable dataset, and thus, characterizing the MCUs. The measured frequencies obtained from the on-chip ROs, also known SMONs, serve as the features for the ML models [1]. The SMONs consist of library cells in series with an overall inverting behavior; thus, the structures oscillate. These SMONs' frequencies are accurately measured during production, providing high-quality features in a stable, fast, and straightforward process. The SMONs measurement is part of the regular production test flow [14], as depicted in Figure 1. Thus, the SMONs data is stored for every produced MCU, possibly leading to millions of unlabeled samples.

An internal counter is used to measure the oscillation frequencies of the SMONs. The high-precision measurement of the SMONs is ensured through a calibration process. Prior to measurement, each SMON undergoes calibration to guarantee the accuracy of the collected data. This calibration process, combined with stable temperature conditions, results in high-quality SMON data, minimizing any systematic errors and variations in the measurement system. The average error rate of less than 0.15%. It is worth noting that this low error rate is achieved under the conditions of the minimum voltage, denoted as $V_{\min}$. This described process is crucial for generating high-quality features for machine learning models and for maintaining the integrity of the dataset. All SMON data are logged for further processing. The SMONs data are acquired both before and after packaging the dies, to analyze the effect of burn-in on the features.

The number of SMONs on a chip depends on the product type, ranging from tens to hundreds. In a legacy product technology family (here referred to as A), the number of SMONs in the different products ($A1, A2 \dots$) remains constant. Whereas in a new and different product technology family (B), the number and the design of the SMONs may change from product to product ($B1, B2 \dots$). For the legacy product family A, each die contains 27 SMONs. In contrast, for the new product family B, the MCU contains a significantly larger number of SMONs, approximately 150, which are distributed across the die.

The target automotive MCUs under study are produced using mature CMOS technology. To capture a wide range of technology behaviors for these MCU products, the manufacturer employs split-lot wafers during the production process. These split lots are specifically manufactured for engineering purposes and serve to simulate slow, typical, and fast dies by adjusting technology parameters. In some of these split lots, the n-MOSFETs are designed to be faster than the p-MOSFETs, and in others, it may be vice versa. The extensive variety of split lots covers nearly all potential technology behaviors, enabling precise predictions and in-depth analysis.

The SMONs on the die can be categorized into five distinct groups based on their structure, identified as INV, NAND, NOR, EXT, and VM:

- INV: These SMONs consist of inverters from the standard library of the design and reveal both n-MOSFET and p-MOSFET behaviors.
- NAND: The NAND SMONs are designed to exhibit a certain n-biased behavior on the die. They are characterized by daisy-chained n-MOSFETs that dominate the NAND cell behavior.
- NOR: The NOR SMONs display a p-biased behavior for these gates, and this bias is further reinforced with the placement of dedicated transistors.
- EXT: The EXT SMONs are more sophisticated and involve replicating specific paths from the design, effectively converting them into ROs. These ROs behave similarly to the selected paths from the original design.
- VM: The VM SMONs are primarily designed and placed on the die for monitoring physical parameter variations across the chip.

Notably, all five groups of SMONs are manufactured using two major types of transistors, which are distinguished by their threshold voltage:

- Regular Threshold Voltage (RegVT) transistors are fast and offer high performance. However, they also contribute significantly to leakage current due to their lower threshold voltage.

- High Threshold Voltage (HVT) transistors have slower switching behavior (low performance) but significantly reduce leakage current.

The choice between RegVT and HVT transistors is made based on the MCU's specifications, either favoring high performance or low power consumption. Typically, the final design consists of a combination of both transistor types, with some design paths containing transistors with homogeneous threshold voltage and others having a mix of HVT and RegVT transistors. The proportion of these transistors varies widely, and it depends on the specific design requirements.

The diverse array of SMON types, combined with the use of various split-lot wafers, provides a solid foundation for investigating the MCU's behavior across all corners of its operational space. This diversity of data is invaluable for data processing, analysis, and optimization during the manufacturing and testing of these automotive MCUs.

C. Data Collection: Label

Labeling MCUs with their $F_{\max}$ is a distinct and time-consuming procedure [1], [14]. It is not integrated into the regular production test flow and involves individual measurements of each MCU using functional test patterns. These patterns are designed to simulate real-world conditions and are executed under worst-case voltage (V_{crit}) and temperature (T_{crit}) settings [27]. During this process, the frequency of each MCU is gradually increased until a functional failure is observed, and the last working frequency $F_{\max}$ is recorded. To ensure the reliability of this data, the process is executed several times (typically five) to confirm that the failing frequency remains consistent. Multiple functional test patterns are employed, resulting in a multi-label dataset. Notably, the most critical pattern, identified as $P_{\min}$, is the one with the lowest $F_{\max}$ value, and this value can vary among different MCUs.

Due to the substantial effort required, the labeling process is performed on only a limited subset of the manufactured devices, leading to a scarcity of labeled data. This scarcity poses a significant challenge for ML-based applications [14], [15]. The quality of the features and label measurements is crucial for the accuracy of ML models and the effectiveness of subsequent MCU performance screening. The accuracy of label measurements is particularly susceptible to uncertainties and noise stemming from minor variations in voltage or temperature conditions, mechanical vibrations, or statistical noise. While SMONs measurements provide high-quality features, label acquisition introduces potential inaccuracies

Furthermore, it's important to acknowledge that every CMOS manufacturing process exhibits particular process variations that typically follow a Gaussian distribution [28]. This means that the majority of devices fall within the expected process variation. However, some devices may reside in the tail region of the distribution and can be considered outliers. Tests conducted in the production flow aim to identify defective devices [29]. Nevertheless, some MCUs with certain defects might escape the testing procedures.

To address these challenges and optimize the information extracted from the available data, various techniques are employed. These include Outlier Detection [16], Active Learning strategies [14], [15], and potentially Transfer Learning. These techniques are crucial for enhancing the quality and reliability of the dataset and ultimately improving the performance of ML models in predicting MCU behavior.

D. Machine Learning

ML is a subfield of artificial intelligence (AI) that focuses on the development of algorithms and models that enable computers to learn from data and make predictions or decisions without explicit programming [30]. The overarching goal is to create systems that can automatically improve their performance over time through experience. ML model can be classified based on several frameworks [30]–[33]. In supervised learning, the algorithm is trained on a labeled dataset (the *training set*), where each input (the *features)* is associated with a corresponding output (the *label*). The model learns to map inputs to outputs, and its performance is evaluated on unseen data (the *test set*). Unsupervised learning involves training models on unlabeled data to discover patterns, relationships, or structures within the dataset. Clustering and dimensionality reduction are common tasks in unsupervised learning. Reinforcement learning focuses on training agents to make sequential decisions by interacting with an environment. The agent receives feedback in the form of rewards or penalties, guiding its learning process. Supervised ML algorithms can be categorized into linear and non-linear based on the mathematical expression between input features and output. Linear Regression fits a linear equation to observed data. Ridge Regression [30] extends this concept by introducing a regularization term to the linear regression objective function, penalizing large coefficients. Neural networks, especially deep neural networks, can model highly complex and non-linear relationships. They consist of layers of interconnected nodes (neurons) and use activation functions to introduce non-linearity into the model.

E. Deep Convolutional Neural Networks

Deep Convolutional Neural Networks (CNNs) have emerged as potent ML models, particularly in the realm of computer vision, where their applications have yielded state-of-the-art results across an extensive spectrum of tasks, spanning both supervised and unsupervised learning [34], [35]. This impressive versatility encompasses image classification, object detection, semantic segmentation, and the extraction of latent features. CNNs architecture is often defined by a multi-layer structure that includes convolutional, pooling, and fully connected layers. These networks are meticulously designed to autonomously acquire hierarchical representations of visual data straight from the unprocessed input. The convolutional layers within a CNN are responsible for the application of a set of learnable filters to the input image, resulting in the generation of feature-maps. Subsequently, these feature-maps undergo non-linear transformations through activation

functions, such as rectified linear units (ReLU [36]), introducing essential non-linearity into the network. The pooling layers downsample the feature maps, reducing spatial dimensions. This process bolsters translation invariance while improving computational efficiency. The final segment of a CNN comprises the fully connected layers, which receive the high-level features extracted by the preceding layers and are primed for executing classification or regression tasks, depending on the final activation function chosen. These layers establish connections between all neurons within one layer to all neurons in the next layer. Furthermore, CNNs find valuable applications in unsupervised learning scenarios, such as those calling for the acquisition of a meaningful representation from unlabeled data. In this context, convolutional autoencoders emerge as a popular choice. Comprising two integral parts—a data encoder network that maps input data to a compressed representation and a data decoder network for reconstructing the input from the encoded representation—these models motivate the network to grasp significant data representations capturing essential features and structural patterns.

Unsupervised CNNs play crucial roles in a variety of tasks, encompassing dimensionality reduction, feature extraction, and image denoising, as previously noted. They also serve as proficient pre-training mechanisms, initializing supervised CNNs to enhance generalization and performance when faced with limited labeled data.

In light of their remarkable capabilities, CNNs predominantly find application in the domain of image analysis due to their exceptional prowess in capturing spatial structures in data. The convolutional kernels within CNNs are intricately designed to harness two key characteristics of input images: local connectivity and spatial locality. The principle of local connectivity dictates that each convolutional kernel operates on a confined region of the input image during the convolution process: the connections are local in space (along width and height), but always extend along the entire depth of the input volume. The assumption of spatial locality (or local correlation) posits that closely related pixels being convolved together yield meaningful feature representations. For instance, a solitary convolutional kernel can adeptly discern edges, textures, shapes, gradients, and other salient features.

CNNs can be adapted to tabular data, as evidenced by recent research [37]. In such instances, tabular data can be effectively transmuted into image-like structures, enabling the seamless application of CNNs. Alternatively, one-dimensional convolutions can be harnessed to process the columns of the dataset, treating each sample with C dimensions as a one-dimensional "image" of size (1, C).

F. Transfer and Semi-Supervised Learning

Transfer learning, a prominent technique in the realm of ML, has emerged as a valuable approach for enhancing performance in a variety of tasks by leveraging knowledge already acquired from solving different problems [20], [38]. This approach proves particularly potent in the context of Deep Neural Networks, where it entails harnessing pre-trained models that have been trained on extensive datasets comprising millions of samples.

These pre-trained models are used as a foundation for a new task or dataset, rather than initiating the training of a Deep Neural Network from scratch. Thus, they are *fine-tuned*. To safeguard the integrity of the acquired representations, the initial layers of the pre-trained network are held in stasis, or frozen, preventing any weight modifications during subsequent fine-tuning. These preserved pre-trained layers assume the role of feature extractors, capturing general patterns and representations from the dataset [39].

The fine-tuning phase introduces further adaptability, as it restricts the training to a final linear layer responsible for bridging the network's extracted features to the specific task labels, be it classification or regression. This strategic partitioning of the network's functionality enables it to adapt seamlessly to the requirements of the task at hand, ensuring a synergistic blend of general knowledge and task-specific expertise.

In parallel, SSL [40], [41] represents a distinctive approach within the domain of ML. SSL exists in the intermediary realm between unsupervised learning, which lacks labeled training data, and supervised learning, which solely relies on labeled data. This approach gains particular relevance when the cost associated with producing labeled data is prohibitively high, while an abundance of unlabeled data is readily available. The key point of SSL is to use both labeled and unlabeled data to derive ML models, in order to enhance the knowledge retrieval from data, and reduce the amount of labeled data needed. It can be interpreted as a Transfer Learning approach in which we aim to transfer knowledge from unlabeled to labeled data.

Various SSL strategies have been developed [21], [40], [42]. The simplest among them, known as "Pseudo-Labeling" unfolds in a three-step process: it initiates by training a learning algorithm on a limited subset of labeled data, subsequently applying the trained model to the unlabeled samples, thereby obtaining approximate labels or predictions. Finally, it refines the model using the entire labeled dataset, which includes both actual and pseudo-labeled data.

For a more robust approach, "Self-Training" [40] emerges as a noteworthy option. This iterative Pseudo-Labeling algorithm selects only the most promising Pseudo-Labeled data based on the probability of correctness or confidence intervals at each iteration. These selected data points augment the labeled set.

Notably, the feasibility of these SSL approaches is contingent upon the underlying model's ability to compute confidence intervals or probabilities of correctness for its predictions, making them particularly suitable for classification models [42], in which they found the majority of application. Even though, some research has been conducted in SSL for regression tasks [43].

The combined concepts of Transfer Learning and Semi-Supervised Learning can enhance the learning ability of ML models, enabling them to acquire knowledge both from unlabeled data and different task: while Transfer Learning leverages pre-trained models to bolster performance in new

domains, Semi-Supervised Learning navigates the challenging landscape between labeled and unlabeled data, enabling cost-effective knowledge acquisition. The underlying synergy between these methods and the adaptability of convolutional neural networks further underscores the depth and breadth of possibilities within the field of machine learning, leading to several benefits, including reducing the amount of required labeled data, speeding up training time, and improving generalization to new tasks.

G. Dimensionality Reduction

There are two primary methods for achieving dimensionality reduction: Feature Selection (FS) and Feature Extraction (FE) [30]. FS [44] involves identifying and selecting the most relevant subset of features from a large pool of potential input variables. The goal is to discard redundant or noisy features that may lead to overfitting or poor generalization. On the other hand, FE aims to reduce the dimensionality of a dataset by extracting the most important features. Principal Component Analysis (PCA) is a popular technique for FE [45]. Unlike FS, FE creates new features by combining the original ones. Methods for FS can be categorized into three main groups: Filter Methods, Wrapper Methods, and Embedded Methods [44]. In particular, Wrapper methods utilize an ML model to evaluate the performance of different feature subsets and select the best one based on error measures computed on a selected test set, often employing cross-validation. Recursive Feature Elimination (RFE) is a widely-used wrapper method [46]. In certain scenarios, applying input space reduction techniques in conjunction with Deep Learning (DL) can offer benefits such as reducing computational complexity and memory requirements of Deep Neural Networks, as well as improving interpretability. It can also be advantageous when dealing with datasets that contain noisy or irrelevant features that can negatively impact the performance of DL models.

IV. Reasoning behind Deep, Semi Supervised and Transfer Learning

Extensive research has been conducted in recent years on predictive models for MCU performance screening [1], [15], [16]. In this scenario, where data scarcity is a concern, techniques such as Active Learning (AL) [15] and Outlier Detection [16] have proven to be useful in creating an informative training set efficiently, thus facilitating the development of robust ML models.

A first step towards training-set size reduction in ML-based MCU performance screening was made in [14], where authors proposed to use AL techniques to select the most informative samples to achieve a higher quality training set. In [16], the authors focused on how anomalies on the labels introduced during measurements in the dataset creation phase can affect ML regression models in the context of MCU performance screening and showed a possible approach to recognize and replace anomalies on the labels from the training set. But these approaches, even if effective in halving the size of the training set, may not be enough: in a context with a very limited amount of samples, having a high-quality dataset helps to create robust models, but the limitation in the number of samples may lead to biased models, not able to well generalize to unseen data.

The very first approach in building a new ML model from scratch is starting from the early phase of obtaining labeled data and training a shallow learning model. However, this process requires waiting until a sufficient amount of labeled data is collected to train a reliable model across the feature domain of interest and evaluate its performance on an appropriate test set. But this phase takes time, in particular in the context of MCU performance screening: considering at least 30 minutes to label one sample [1], obtaining a suitable training set for learning the ML model would require 63 days of continuous labeling for about 3000 samples, with no missing values and outliers. If we consider possible outliers and noise in the label measurements and the fact that some samples may be dropped because of that (about 1 over 4 [1]), this time increases by a factor of 1.25. This duration does not even consider the feature engineering phase, which is often the most time-consuming aspect of developing a predictive model and may take additional months.

In a context of scarce data, traditional shallow learning models (SVM, Ridge Regression, ElasticNet etc.) tend to outperform DL models, due to the increased complexity of the latter which often leads to over-fitting. However, shallow learning requires precise feature engineering as a preprocessing step to build better models. This involves supervising the learning process by incorporating domain knowledge, such as applying polynomial transformations to raw features before feeding them into a Linear Regression model [14]. In contrast, deep learning models do not necessarily require this step or do not consider it as crucial. DL models raised a huge amount of interest thanks to their supremacy in terms of accuracy when applied to big data. DL models are able to learn how to represent the data by hierarchical concepts in a nested way, with each concept defined in relation to simpler concepts [39], and more abstract representations computed in terms of less abstract ones, moving from input to output layers. They are able to extract features and learn how to find the best data representation autonomously, by manipulating input features, rather than relying on a-priori hand-crafted features (as in shallow learning). Also, the architecture of the networks can be tuned without any limit.

However, when dealing with limited labeled data, the aforementioned DL models cannot be applied. This situation commonly arises when manufacturing a new product, and only few data are available. But DL models are flexible: it is possible to pre-train a model on a certain training set (often composed of a huge number of samples, millions) and then fine-tune it on a related dataset, in which we want to solve a similar task. This approach is the foundation of Transfer Learning (see Section III-F). Instead of waiting to collect a sufficient number of labeled data, Semi-Supervised and Transfer Learning can be employed. In particular, Transfer Learning enables using only a portion of the architecture of a DL model (e.g., the

first n layers) to extract a relevant feature embedding, using the models as a features transformers or encoders.

When an MCU product reaches large-scale mass production, millions of (unlabeled) data are produced and can be, in principle, used to enhance the knowledge retrieved by successive data analysis. Also, when a new product is engineered and then deployed, typically, data and models from different or previous products are still available. Even if applying the very same model trained on legacy products to new ones is infeasible as it leads to significant prediction errors, it is also true that the aforementioned models have learned valuable information from the data, which can be utilized to derive new models for the new product. In particular, the latent space discovered by deep neural networks can remain useful, making these models suitable as feature extractors. Only the final layer, responsible for linking the extracted features to the target variable, needs to be trained. If the features extracted by deep neural networks are sufficiently general, only a few samples would be required to train these final supervised models.

For all these reasons, one can naturally think about switching to DL models, leveraging SSL and Transfer Learning if only a small amount of labeled data is available. The rationale behind using Transfer Learning becomes evident: save time in the data collection phase by leveraging knowledge acquired from similar (though not identical) domains. This is accomplished by transforming the input features using a pre-trained network and training a shallow supervised linear model that connects these features to the target variable. This permits to develop models that are transferable across different MCU products, providing a foundation that can be applied across various contexts. These models establish a robust baseline for further investigation and refinement.

V. Proposed Approach

The approach followed in this work aims at extracting a feature representation of the input data (i.e. the SMON values) by means of DL models, used as transformers. We will first use Semi-Supervised Techniques to pre-train the DL models on both unlabeled data from production lots (available in millions of samples) and labeled data from corner lots, to learn a latent space in which project SMON values. In this way, DL models are used to compute non-linear features interaction. By using the transformed SMONs values, solving the successive supervised task (i.e. linking the $F_{\max}$ label to the transformed SMON values) should be possible just with a linear model (Ridge Regression, as an example). The final goals are:

- Increasing the prediction ability of the models with respect to the classical shallow learning approach and manual feature engineering.
- Reducing the amount of labeled data needed to train these models, with a consequent reduction in the dataset creation phase (i.e. labeling the samples).

Once that deep feature extractors are pre-trained on millions of unlabeled data of a certain product, $A1$, it is possible to use them even on different products with the same SMONs set (here called $A2$), or even totally different SMONs set (here referred to as $B1$). Thus, the goal of this work is to implement a transfer learning framework of two types:

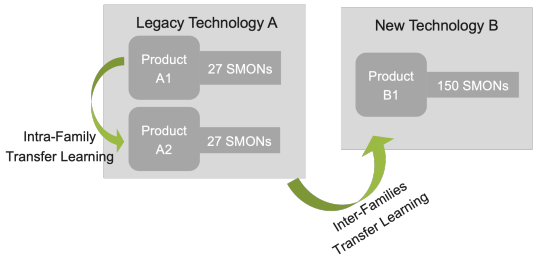

Fig. 2. The proposed Inter and Intra families transfer learning protocol.

- Intra-Family: we train the deep learning models on product $A1$ of product family A, and we try to use the same model as a feature extractor for a different product, $A2$, of the same family A. Products in the same family have the same SMONs set.
- Inter-Families: we train the deep learning models on product $A1$ of product family A, and we try to use the same model as a feature extractor for a different product, $B1$, of a different family, B. Products of different families have different SMONs set. Proper countermeasures to adapt the different input spaces have to be taken.

The adopted Transfer Learning framework is shown in Fig. 2

The deep learning feature extractor creation phase consists of four steps:

1) We train the deep learning models to perform a certain (alternative) task T_i using the unlabeled data from *A1*.
2) We (optionally) fine-tune the deep models using the labeled data, to adapt the parameters of the convolutional kernels to the actual labeled data distribution: using the model trained on the unlabeled data as a starting point (i.e., we use the pre-trained layers' weights as initialization), we can run again the training optimization on the labeled data, in a warm-start fashion.
3) We drop the last layer of the DL models (that actually link non-linear features interaction, computed in the previous layers, to the the task chosen in step 1). Every model is now used as a data-transformer able to project the data from the original space into a new one in which solving the original performance prediction task is easier.
4) Finally, we train a linear model (for example, a Ridge Regression) using the features extracted in the previous step as inputs. Such a model will combine them to solve the performance prediction task (in the case of the Ridge Regression, by means of an L2-regularized Mean Squared Error loss).

An example of the proposed approach is shown in Fig. 3.

The step 2 is optional since it depends on the size of the labeled dataset. Too few samples lead to overfitting, causing the network to not be able to generalize on new unseen

samples, performing worse than the feature extractor with no fine-tuning. However, if the representation learned by the network on *A1* is general enough, this step does not lead to an enormous improvement in the prediction performances. In the absence of fine-tuning, the deep models are used "as-it-is", in the sense that the layers are frozen, and no parameters are changed. For step 1 of the proposed approach, we need to pre-tain the DL models on and alternate task T_i on the unlabeled data. We selected three popular pre-training strategies, described in the following subsections.

A. Pseudo-labeling

We first build a supervised ML model as a prototype, using the actual available labeled data. We then apply this model to the unlabeled data, obtaining a pseudo-label for each unlabeled sample. The goal is to create an augmented training set $\hat{X_u} = \{(x_{u_1}, \tilde{y}_{u_1}), (x_{u_2}, \tilde{y}_{u_2}), \ldots, (x_{u_n}, \tilde{y}_{u_n})\}$. Each pseudo-label $\tilde{y}_{u_i}$ is directly correlated with the final task we aim to solve (i.e., performance prediction). A supervised DL model is then trained on $\hat{X_u}$. This model will acquire knowledge on how to manipulate SMON values to generate features useful to the performance prediction task. The model is then fine-tuned on the actual labeled training set. The model is not "blind" to the problem we aim to solve: it uses both labeled and pseud-labeled data to enhance knowledge retrieval. We expect that models built upon the feature extracted by using pseudo-labeling will achieve higher accuracy with respect to unsupervised DL feature extractor.

B. Auto-Encoder

The task is based on trying to reconstruct the initial input. We define the training set as $\hat{X_u} = \{(x_{u_1}, x_{u_1}), (x_{u_2}, x_{u_2}), \ldots, (x_{u_n}, x_{u_n})\}$, so the input and the output of the deep ML model are the same. The model can be divided into two parts: the encoder projects the data into an alternate space, $\widetilde{X}$ and the decoder learns how to recover the original data, X, from the alternate space $\widetilde{X}$ (see Fig. 3). This task is totally unsupervised because we are using only the SMON values X of the unlabeled set. At the end of the process, we drop the whole decoder, using only the encoder as a features extractor (as depicted in Fig. 3). The features manipulation, $\widetilde{X}$ are then linked to $F_{\max}$ by a linear regression model. This task is "blind" and not aware of the actual performance prediction task we aim to solve, and the features extracted will be a more general non-linear manipulation of the SMONs input features.

C. Denoising Auto-Encoder

Similar to the previous task, here the inputs of the model are a noisy version of the data (with a superimposed Gaussian noise). The training set is thus $\hat{X_u} = \{(\tilde{x}_{u_1}, x_{u_1}), (\tilde{x}_{u_2}, x_{u_2}), \ldots, (\tilde{x}_{u_n}, x_{u_n})\}$, in which $\tilde{x}_{u_i}$ is a corrupted version of x_{u_i}. The model should learn how to recover data from their noisy version. The obtained model can be again divided into an encoder and a decoder. Also, this task is totally unsupervised and problem-unaware. At the end of the process, we drop the decoder and use only the encoder as features extractor.

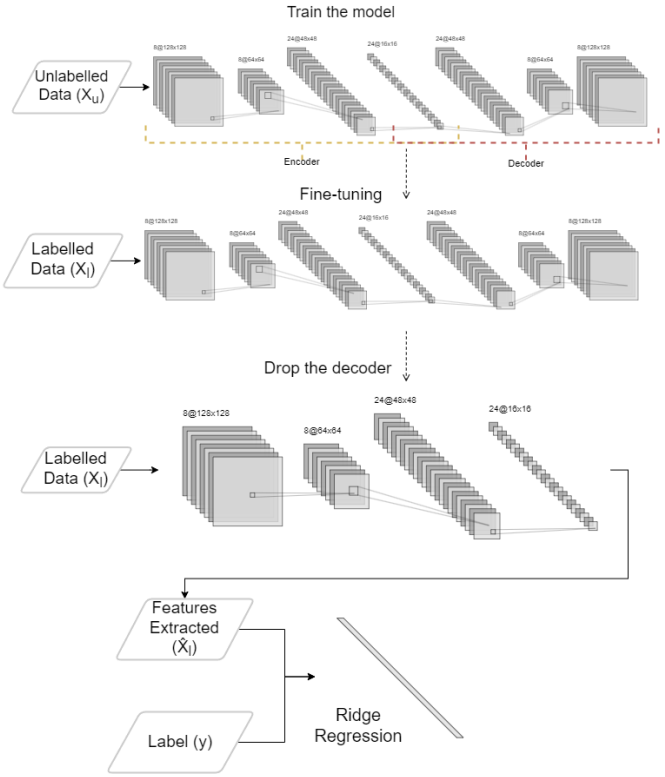

Fig. 3. Example of training a deep convolutional feature extractor in a Semi-Supervised fashion, using the Auto-Encoder alternate task. We first fed the deep model with unlabeled data. We then use it as a transformer for labeled data (for the same or different product). Finally, we train only a Linear Model to link feature embeddings and labels.

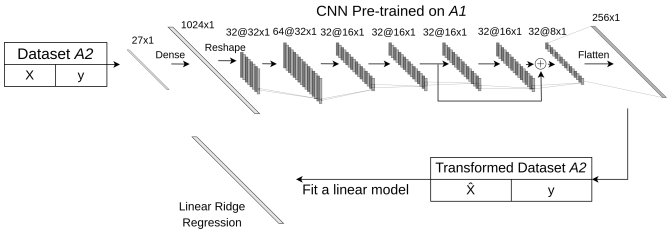

Fig. 4. Example of the proposed transfer learning approach between *A1* and *A2* product using the PL-CNN. We first transform the *A2* dataset with the pre-trained network. The network is frozen, and no convolutional layers are updated. We then train a linear regression to link feature embeddings from *A2* and $F_{\max}$ labels.

VI. Experimental Evaluation

A. Experimental Setup

The proposed methodology has been validated on a dataset composed of 2986 labeled *A1*-devices and 1015 labeled *A2*-devices from product family *A*. The deep neural networks were pre-trained by using 1,496,248 *A1*-unlabeled samples from production lots. *A1* and *A2* devices, coming from the same product family, are equipped with the same SMONs on board. Thus, for both products, we have 27 SMONs. For

A1, 10 labels are available, measuring the $F_{\max}$ for different functional test patterns. For *A2*, 20 labels are available. For both products, the final performance of the devices and thus the target label of our model (the maximum operating frequency) is the artificial label $P_{\min}$ (the minimum among the available labels, see Section III-C).

For the product family B, the dataset is composed of 2002 labeled samples of product $B1$. In the experiments, 150 SMONs of the product $B1$ were considered.

Features are standardized by removing the mean and scaling to unit variance, as a pre-processing step.

A preliminary Outlier detection technique was applied [16]. We used the Interquartile Range (IQR) method [47], applying it on the input (SMONs) space. Samples that fall outside a specific range $[lb, ub]$ are considered outliers and removed from the dataset. It involves to calculate Q_1 and Q_3, to compute the IQR as $Q_3 - Q_1$, and finally to determine the lower bound (lb) and upper bound (ub) as:

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

As the baseline comparison model, we used a Ridge Regression with PCA with 14 components and polynomial transformation of the input SMONs values [14] (namely, Poly Ridge, PLR or PCA Poly Ridge). Previous research on $A1$ [14] showed that SMONs-$F_{\max}$ relation is not exactly linear, with Polynomial models better fit this relation. However, we included a comparison with linear models (Ridge Regression, called L. Ridge and Linear Regression, called L.Reg), in addition to Random Forest [1], [3] and KNN-5 [3].

As a baseline, we first applied the very same Poly Ridge model trained on *A1* directly on *A2*, with no fine-tuning. This shows the impossibility of re-using the very same model on different products and the necessity of updating it. Next, we trained the baseline Poly Ridge only on samples from *A2*, without considering previous knowledge.

Once the deep feature extraction models are derived from $A1$, they are applied on the new data from *A2* and $B1$ obtaining the transformed datasets $D^{A2} = (\hat{X}^{A2}, y^{A2})$ and $D^{B1} = (\hat{X}^{B1}, y^{B1})$, in which $\hat{X}^{A2}$ is the projection of X^{A2} in the latent space learned by the Neural Network. The same holds for $\hat{X}^{B1}$. At this point, it is possible to train a simple Linear Regression model (in our case, a Ridge Regression) on D^{A2} and D^{B1}, to learn how to map the transformed SMONs values to the actual $F_{\max}$. In the test phase of these models on *A2* and on *B1*, the SMON values are first transformed with the deep feature extractor. Then, the Linear Ridge Regressor is applied to predict operating frequency $F_{\max}$. The order of application of the DL models $A1 - A2 - B2$ follows data availability and similarity among products. If data permit this, it could be possible to change the order of training, taking care of *increasing* the dimensionality of the features to adapt A-SMONs set to B-models (instead of *decreasing* it, as done in this paper).

We used a 5 train-test split with proportion 75%-25%, generated by different random states, to build learning curves, to eventually fine-tune the deep feature extractor, and to evaluate the models: thus, each statistical prediction performance is the mean of 5 values computed on 5 different training/test splits of the dataset, avoiding biased prediction error estimation. Results are presented in terms of normalized Root Mean Square Error (nRMSE), normalized Mean Absolute Error (nMAE), Learning Curves, the Area Under the nRMSE Learning Curve (AUC-nRMSE), and Guardband G. RMSE and MAE are popular regression performance indexes [48], but normalized by the mean value of $F_{\max}$ in the test set, i.e. $nRMSE = RMSE(y_{true}, y_{pred})/mean(y_{true})$ and $nMAE = MAE(y_{true}, y_{pred})/mean(y_{true})$, to obtain a percentage of the error. The learning curve plots correlate the training set size with the generalization capabilities of a model. At each point of the curve, on the x-axis we have the number of samples used to train the model and on the y-axis a measure of prediction performance of the model on the test set (in our case, the nRMSE). The learning curves were created by extracting (for each point x-y) a random sample of the training set of increasing size. These subsets were used to train the final linear model and to eventually fine-tune the deep models. The AUC-nRMSE value indicates the generalization capability of a model: the lower this value, the better the ability of the model to generalize with fewer labeled samples. We computed AUC-nRMSE values for the first 20 points of the learning curves, where only a few samples are available (until 300 labeled samples for $B1$, 600 for $A1$, and 170 for $A2$).

To accommodate potential errors and uncertainties in statistical predictions, a risk-based guardband is necessary [14], [49]. This error guardband (G) surrounds the specified limit and serves as a buffer beyond acceptable product limits. By implementing the guardband, manufacturers can guarantee that even slight variations or uncertainties do not compromise the product's quality standards or reliability. Practically speaking, supposing that the screening frequency is f_{screen}, the effect of G is to increase the threshold for the pass/fail screening from f_{screen} to $f_{screen} + G$. Since G impacts the production yield [14], [49], [50], it should be minimized. We can compute G on a test set with true frequencies y, predicted frequencies $\hat{y}$, and errors $e = y - \hat{y}$ as:

$$G = \mu_e + k\sigma_e$$

μ_e and σ_e are the mean and the standard deviation of the error distribution and k is a parameter that permits the choice of the defects' level in ppm. $k = 5.2$ is an approximation for 0.1 ppm, but we used a more stringent value ($k = 6$). G will be expressed in percent of the actual $F_{\max}$ specification, as described in the datasheet. As an additional evaluation metric, we will use the number of samples needed to reach the 2% of nRMSE on the test set. A 2% nRMSE typically signifies a favorable level of error, allowing us to achieve a satisfactory margin of safety G and consequently a desirable level of predictive performance screening yield [50].

All experiments were performed in Python using PyTorch tools for the DL models. Experiments run on a server equipped with an Intel® Core™ i9-9900K CPU @3.60GHz x 16, 32GB of RAM, and an Nvidia® 2080 TI GPU.

B. Inter-Family Transfer Learning setup

To implement the inter-family transfer learning protocol, a first pre-processing step is needed to project the input space of product $B1$ (of dimensionality of 150 SMONs) to the number of inputs of the neural network (n, in our case with $n = 27$). Several techniques are investigated:

- PCA-Projection: using PCA to match the SMONs spaces
- Random SMONs Sampling: a random choice of n SMONs (namely, Dummy-Feature Selection CNN, shortly Dummy-FS CNN or D-FS CNN). SMONs are usually highly linearly correlated with each other, thus a random choice of them is reasonable. This approach is unbiased,requiring no additional information about SMONs type and technology.
- Repeated SMONs Sampling: This approach extends the previous method by iteratively sampling n SMONs from the pool of 150 available, utilizing them to train an ensemble of network. We created an ensemble of 10-Dummy CNN models. Randomly sampling various inputs, permits exploring different areas of the SMONs input space, allowing each model in the ensemble to learn distinct feature interactions. Also, Random Features Sampling has been demonstrated to effectively reduce variance in predictions and enhance accuracy in classification tasks [51]. The final prediction is the average of the 10 outcomes (hard-voting mechanism). This model is referred to as Dummy-Feature Selection Voting CNN, abbreviated as Dummy-FS Voting CNN or D-FS V-CNN.
- Best SMONs: An FS strategy based on choosing the best n SMONs. If test engineers have a piece of information on which are the best SMONs for the performance prediction task (the ones the most linked to the performance of the devices), they can manually select these to build the ML models. We estimated the SMONs importance by applying an RFE-based approach repeatedly on the whole labeled training set [17]. We built a SMONs ranking, and we selected the best 27 SMONs. We called this BEST-FS. This should be the "gold standard" and should be considered only as a comparison because the importance of each SMON was computed on the whole labeled set.
- Correlation Approach: Basing on [3], we identified the n SMONs with the highest Pearson correlation coefficient with the target $F_{\max}$. We called this Corr-FS.

In general, the reduction in the complexity of the model (in terms of parameters/features analyzed) can also lead to a reduction in the number of labeled samples to achieve a certain generalization error. So, as a comparison, we will show experiments of applying the aforementioned Feature Reduction techniques (PCA, Best-FS) to the Polynomial Ridge, too. The architectures and the training setup for the CNNs used are described in the following section Section VI-C.

C. Deep-Learning Models

We trained and tested the following DL models on $A1$ as described in Section V, but only the best will be chosen to be fed with $A2$ and $B1$ data for the Transfer Learning protocol

1) A Soft-Ordering 1D-CNN with skip connection (namely CNN with Pseudo-Labeling or PL-CNN): 1-dimensional CNN with a fully connected layer as the first layer, with CELU (Continuously Differentiable Exponential Linear Units) activation function [36]. This first layer projects the features into a higher dimensional space using a non-linear combination of the SMONs(see Fig. 4). The reason is given: since tabular datasets are not spatially correlated (as in images), CNNs would not be able to express their ability to catch local interactions between features. But if we re-order the tabular features, a 1-D convolutional layer may extract some relevant interactions between them [52]. The output of the first layer is then reshaped into image-like samples of dimension $(H, 1, C)$ (height H, length 1, and C channels). Each of these corresponds to a group of H ordered features, and we have C groups with different orderings. The output of the network is then flattened again before going into the final supervised model.
2) Fully-connected AE (AE-FC): Fully connected layers that perform feature space augmentation. It does not present a bottleneck but projects the data into a higher dimensional space.
3) Denoising fully-connected AE (DAE-FC): as above, but the input of the training procedure is a corrupted version of the original data.
4) Convolutional AE (AE-CNNE): AE with soft-ordering, 1D convolutional encoder, and de-convolutional decoder. The encoder performs feature space augmentation. The output of the encoder is flattened before going into the final supervised model (see Fig. 4).
5) Denoising Convolutional AE (DAE-CNN): as above, but trained on a corrupted version of the data.

Each convolutional layer is preceded by a Batch Normalization layer and a Dropout layer. The setup for training the DL models on *A1* is here described: 85% of the training data set was used to train the models, which are validated on the remaining 15%. The training procedure runs for a maximum of 100 steps (*epochs*), or until we are not able to increase the performance on the validation set with respect to the last 10 epochs (*early stopping*, [53]). We used a Stochastic Gradient Descent (SGD) optimizer [53] to tune the weights of each layer, with learning rate lr equal to 1×10^{-4} and momentum equal to 0.7 [53]. The lr decreases on loss plateau until 1×10^{-5}. The very same approach was used to eventually fine-tune the deep models (fine-tuned models were clearly indicated in the tables in Section VII-B). In the absence of DNN fine-tuning, the only layers that are updated are the Batch Normalizations, to adapt them to new data distribution ($A2$ and $B1$).

VII. Experimental Results

In the following section, we will present the results of a deep-learning-based MCU performance prediction. We will first present the results of applying Semi-Supervised techniques to derive deep feature extractor models on $A1$ data.

We will show comparisons between these and the baseline shallow-learning model (Polynomial Ridge), and the gain in both prediction performance (percentage of error) and the amount of labeled samples needed to reach baseline performance. We then will choose the best deep feature extractor models to implement the transfer learning framework between $A1$ and $A2$ devices. In the end, we will show the results of applying inter-families transfer learning on $B1$ samples.

A. Semi-Supervised Learning on $A1$ product

Experiments showed that almost all the DL models can extract relevant information from the unlabeled data, with final prediction errors aligned to baseline Polynomial Ridge (Table I and Fig. 5). Linear models failed at fitting the data, with the highest prediction error and AUC. For this reason, these are not shown in Fig. 5. Non-linear models such as KNN-5 and RandomForest (RF) reached decent prediction errors, but cannot stick with the baseline.

Even the features extracted by a simple fully connected auto-encoder are appropriate to our goals.

The CNN-PL seems to be the best model for two reasons: the first reason is that the achieved nRMSE on the test set and the computed guardband are significantly lower with respect to the baseline approach (1.48% vs 1.57% nRMSE, 9.43% vs 10.04% G, see Table I). Since the guardband affects the production yield, it is necessary to maintain it as lower as possible. The second reason is that with just a fraction of the labeled data (89 for PL-CNN), the nRMSE drops and stably remains under the 2% of error: the learning curve presents a plateau, where the error decreases very slowly (see Fig. 5). This neural network has effectively found useful features for the underlying regression problem, and it is able to generalize to new unseen data even with a small training set. With just 500 labeled samples, we were able to compete with the performances of the baseline Polynomial Ridge model (1.57% of nRMSE), obtained with the whole available training set (2986 samples). The reduction in the number of labeled data permits decreasing the effort during the MCU characterization phase and data collection, thus reducing the time required for acquiring a proper dataset for ML models. Labeling a sample requires at least 30 min. To reach 1.57% nRMSE, with the baseline Poly Ridge model we would require $30 \text{ min} \cdot 2986 \text{ samples} = 89,580 \text{ min} \approx 63$ days of continuous labeling. With CNN-PL, the dataset creation requires less than $\frac{1}{6}$ of the time ($\approx$ 10 days) This holds even for the other DL models: we reached the target prediction error with fewer samples, even if the reduction in the training set size is not significant (Table I). However, with all the DL models we are able to reach errors below the 2% (dotted line in Fig. 5) with just tens of data (89 for PL-CNN, AE-FC and DAE-FC, 119 for AE-CNN, and 149 for DAE-CNN). The reason the PL-CNN seems to be the best approach may be due to the awareness of the performance prediction task. Consequently, the features extracted using this method are directly finalized to solve the actual problem, while the other methods perform a more general feature manipulation. For this reason, this will be the main feature extractor for the successive experiments on *A2* and *B1* products. To compare it with another deep model, we chose the AE-CNN, and we applied it on *A2*.

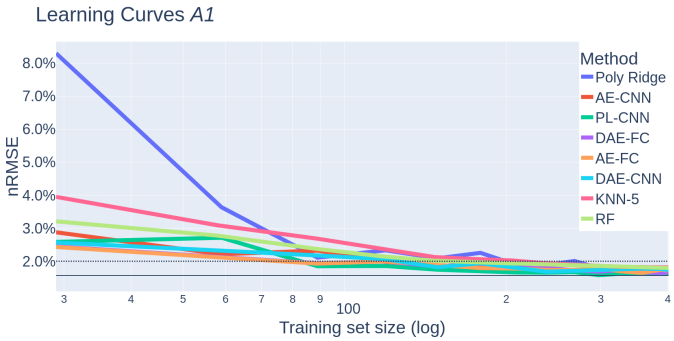

Fig. 5. Learning curves for different deep and shallow models. The x-axes is the number of training samples (log scale, truncated at 400 samples) while the y-axes is the nRMSE on the test set. The upper dotted line is the 2% of nRMSE. The lower line is the baseline Poly Ridge prediction error, obtained with the whole training set (1.57% nRMSE).

TABLE I
PREDICTION PERFORMANCE METRICS OF DEEP FEATURE EXTRACTORS ON PRODUCT *A1*

Model	nRMSE	nMAE	G	Samples to 1.57% nRMSE	AUC
Poly Ridge [14]	1.57%	1.17%	10.04%	2,986	12.1
L. Ridge [1], [3]	2.60%	1.95%	16.54%	–	15.1
L. Reg. [3]	2.60%	1.95%	16.55%	–	19.1
RF [1], [3]	1.63%	1.22%	10.31%	–	11.0
KNN-5 [3]	1.69%	1.27%	10.31%	–	11.5
PL-CNN	**1.48%**	**1.09%**	**9.43%**	**477**	**9.83**
AE-FC	1.56%	1.15%	9.96%	1,702	10.1
AE-CNNE	1.52%	1.13%	9.70%	1,971	10.4
DAE-FC	1.55%	1.14%	9.95%	1,702	10.1
DAE-CNN	1.54%	1.14%	9.85%	1,822	10.4

B. Transfer Learning on $A2$ dataset

As shown in Table II, applying a model trained on *A1* directly on *A2* leads to an enormous prediction error (first row, 33.93% nRMSE). The coefficient of the model should be adapted to the new product. The re-training of the shallow Poly Ridge model on *A2* is effective since we can reach good accuracy (1.54% mean nRMSE, with a standard deviation of 0.11%, in 5 folds) with all the samples in the training set. But to do this, we first need to have a training size big enough. The final prediction performances reached by deploying deep feature extractor are better in terms of nRMSE and guardband (1.51% mean nRMSE with standard deviation 0.13% vs 1.54% mean nRMSE and standard deviation of 0.11%, 11.01% mean guardband with standard deviation 1.05% vs 11.19% mean guardband with standard deviation 0.85%, with the model PL-CNN, Table II). But these good results are reached with a fraction of the labeled data: 227 samples are needed to obtain 1.54% of nRMSE with the PL-CNN, and 16 to 2% (Fig. 6). Experiments showed that the feature extractor DL models, trained on *A1*, can extract relevant information from *A2*. This is true even without fine-tuning (FT, in the plots) the intermediate

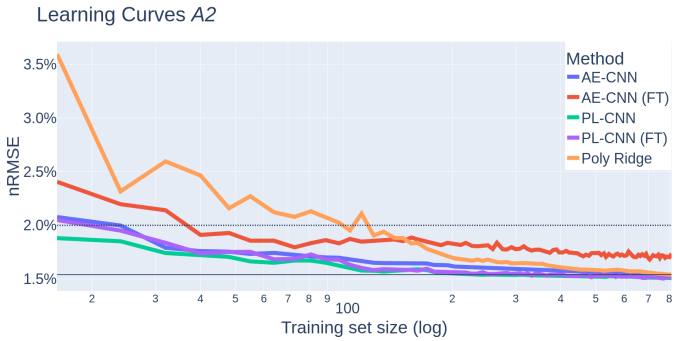

Fig. 6. Learning curves for different models on *A2* product. The x-axes is the number of training samples (log scale) while the y-axes is the nRMSE computed on the test set. The upper dotted horizontal line is 2% of nRMSE. The lower horizontal line is 1.54% nRMSE, the final error obtained by the Polynomial Ridge. With Transfer Learning and Deep Feature extractors, we are able to obtain lower prediction error with a smaller training set.

TABLE II
PREDICTION PERFORMANCE METRICS OF INTRA-FAMILY TRANSFER LEARNING ON PRODUCT *A2*

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

convolutional layers, meaning that the feature manipulation performed by the neural networks is general enough, and can be successfully applied to the different products. Fine-tuning the intermediate layers model does not lead to improvement, and this may be due to the low amount of training data from *A2*, not sufficient to properly update the entire networks (that have a number of parameters on ten-thousands of magnitude), leading to over-fitting. The reduction in the number of labeled data permits decreasing the effort during the MCU characterization phase and data collection, thus reducing the time required for acquiring a proper dataset for ML models. As we stated in the previous section, labeling a sample requires at least 30 min. By using the PL-CNN, building and deploying ML-predictive model for the operating frequency of a new MCU requires just 30 min $\cdot$ 16 samples $=$ 480 min $=$ 8 hours. Also, no additional feature engineering phase is required, due to the ability of the NNs to extract features autonomously.

C. Transfer Learning on $B1$ dataset

Finally, we implemented the inter-families transfer learning protocol, trying to transfer knowledge from A to B. We used the PL-CNN because of its superior performance in the previous tasks. We compared it with Poly Ridge, KNN-5 [3] and Random Forest [1], [3].

Figure 7 shows the learning curve for products $B1$. Even in this case, concerning the baseline Poly Ridge, the CNN pre-trained on millions of samples can obtain lower prediction error with the need for fewer labeled samples.

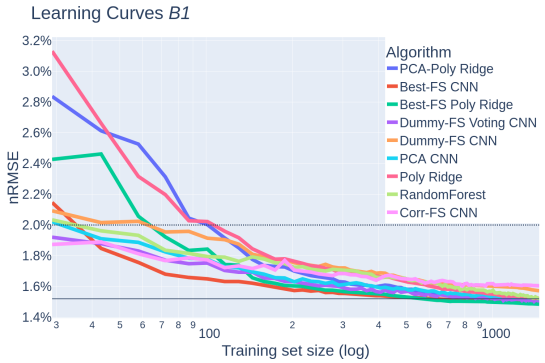

Fig. 7. Learning curves for different models on product $B1$. The x-axes is the number of training samples (log scale) while the y-axes is the nRMSE computed on the test set. The upper dotted horizontal line is the 2% of nRMSE. The lower horizontal is the 1.52% nRMSE, the final error obtained by the Polynomial Ridge. Simplifying the model (with FS) and the use of Transfer Learning can reduce the number of labeled samples needed for models' training.

TABLE III
AVERAGE RESULTS ON $B1$ FOR 5 DIFFERENT TRAINING-TEST SPLITS

Model	nRMSE	nMAE	G	Samples to 2% nRMSE	AUC nRMSE
B-FS PLR[1]	1.49%	1.15%	9.04%	63	4.82
PCA-PLR [1], [14]	1.50%	1.16%	9.16%	106	5.24
PLR	1.52%	1.17%	9.26%	101	5.35
KNN-5 [3]	1.65%	1.27%	10.07%	87	5.36
RF [1], [3]	1.53%	1.17%	9.30%	35	4.91
B-FS CNN[1]	1.49%	1.16%	9.08%	35	4.52
D-FS V-CNN	**1.50%**	**1.16%**	**9.17%**	**29**	**4.66**
PCA CNN	1.52%	1.17%	9.25%	31	4.73
D-FS CNN	1.57%	1.21%	9.55%	63	5.06
Corr-FS CNN	1.60%	1.24%	9.76%	29	4.8

Applying feature reduction to Poly Ridge (PCA, BEST-FS, see Section VI-A), is beneficial. BEST-FS Poly Ridge has a better learning curve if compared to Poly Ridge with and without PCA. In general, simplifying the model, reducing its complexity in terms of the number of parameters, helps for a reduction in the number of labeled samples needed to reach a certain generalization error. With a simpler model, the risk of overfitting is reduced when the number of training samples is small. Among the other models, CNNs achieve the lowest error in the first stage of the learning curve. These are very good to generalize with a limited training set. The best performance can be reached with the ensemble of Dummy-CNN: having an ensemble of models helps reduce the variance of the model itself and thus, increases the prediction performance [54]. In the initial phase of the learning curve, Random Forest shows comparable performance to CNNs, but ultimately, CNNs exhibit lower prediction errors when trained with the entire dataset.

The obtained CNNs are general enough to transfer the knowledge from product A to B. They found a feature representation that can be extended across product families.

VIII. Conclusions

We presented a deep learning-based framework for optimizing the MCU performance screening with ML techniques. This approach, based on Semi-Supervised and Transfer learning, permitted us both to reduce the prediction error of our models and decrease the number of labeled samples needed to build prediction models up to a factor of six for the product family A. Using the available unlabeled samples, we were able to build deep neural networks that act as feature extractors, by projecting data into a higher dimensional space, in which a simple linear Ridge Regressor can predict the performances of the devices. The reduction in the prediction error permits reaching lower guardband G, thus increasing the process yield, since the number of good devices incorrectly discarded would be reduced. The deep learning models we derived, pre-trained on a legacy product *A1*, can serve as a base to develop successive models for MCU products of the same or different family, with the same or different SMONs. This enables the use of Transfer Learning on a new product (*A2*), or to a new family (B). Transferring knowledge between products with the use of Deep Feature extractors helps in reducing the training set size. The reduction in the number of labeled data permits decreasing the effort during the MCU characterization phase and data collection, thus reducing the time required for acquiring a proper dataset for ML models. If carefully chosen techniques are used, just tens of samples are needed to build and deploy an ML-predictive model for MCU performance screening. For product (*A2*), with deep feature extractors, we need 1 over 4 of the labeled samples needed to obtain the same amount of error of a baseline Poly Ridge, and just units of samples (about 15) to have satisfactory prediction error and guardband (2% of nRMSE, 9%-10% of guardband). For product (*B1*), less than 30 samples are needed to have satisfactory prediction error and guardband.

The developed SSL/Transfer Learning framework is general and can be deployed in almost every scenario with a huge amount of unlabeled data, such as MCU performance screening or alternate tests. Also, the feature extraction step (thus, the transfer learning approach) presented in this paper can be used in other scenarios in which pre-trained models are available, to re-use previous knowledge on different but similar domains, and when labeled samples are hard to obtain.

References

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

[2] J. Zeng *et al.*, "On correlating structural tests with functional tests for speed binning of high performance design," in *IEEE International Test Conference (ITC)*, 2004.

[3] J. Chen *et al.*, "Data learning techniques and methodology for fmax prediction," in *IEEE International Test Conference (ITC)*, 2009.

[4] J. Chen *et al.*, "Selecting the most relevant structural fmax for system fmax correlation," in *28th VLSI Test Symposium (VTS)*, 2010.

[1]Best Feature Selection (B-FS) should be considered as a "gold standard" for comparison only: the importance of each SMONs was computed on the whole labeled set.

[5] S.-P. Mu *et al.*, "Statistical framework and built-in self-speed-binning system for speed binning using on-chip ring oscillators," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2016.

[6] K. von Arnim *et al.*, "An effective switching current methodology to predict the performance of complex digital circuits," in *IEEE International Electron Devices Meeting (IEDM)*, 2007.

[7] T. B. Chan *et al.*, "DDRO: A novel performance monitoring methodology based on design-dependent ring oscillators," in *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, May 2012.

[8] N. Bellarmino *et al.*, "Enabling Inter-Product Transfer Learning on MCU Performance Screening," in *IEEE Asian Test Symposium (ATS)*, 2023.

[9] N. Bellarmino *et al.*, "Semi-Supervised Deep Learning for Microcontroller Performance Screening," in *IEEE European Test Symposium (ETS)*, 2023.

[10] L.-C. Wang, "Experience of data analytics in eda and test—principles, promises, and challenges," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2017.

[11] L.-C. Wang, "An autonomous system view to apply machine learning," in *IEEE International Test Conference (ITC)*, 2018.

[12] G. Sannena *et al.*, "Low overhead warning flip-flop based on charge sharing for timing slack monitoring," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2018.

[13] M. Sadi *et al.*, "SoC Speed Binning Using Machine Learning and On-Chip Slack Sensors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2017.

[14] N. Bellarmino *et al.*, "A Multi-Label Active Learning Framework for Microcontroller Performance Screening," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2023.

[15] N. Bellarmino *et al.*, "Exploiting active learning for microcontroller performance prediction," in *IEEE European Test Symposium (ETS)*, 2021.

[16] N. Bellarmino *et al.*, "Microcontroller Performance Screening: Optimizing the Characterization in the Presence of Anomalous and Noisy Data," in *IEEE International Symposium on On-Line Testing and Robust System (IOLTS)*, 2022.

[17] N. Bellarmino *et al.*, "Feature Selection for Cost Reduction In MCU Performance Screening," in *IEEE 24th Latin American Test Symposium (LATS)*, 2023.

[18] S. J. Pan *et al.*, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, 2010.

[19] C. Tan *et al.*, "A Survey on Deep Transfer Learning," *27th International Conference on Artificial Neural Networks (ICANN)*, 2018.

[20] F. Zhuang *et al.*, "A Comprehensive Survey on Transfer Learning," *Computing Research Repository (CoRR)*, 2019.

[21] X. Yang *et al.*, "A survey on deep semi-supervised learning," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[22] J. Donahue *et al.*, "Decaf: A deep convolutional activation feature for generic visual recognition," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014.

[23] T. Hospedales *et al.*, "Meta-Learning in Neural Networks: A Survey," *arXiv*, 2020.

[24] H. Peng, "A comprehensive overview and survey of recent advances in meta-learning," *arXiv*, 2020.

[25] T. Ruokonen *et al.*, *Fault Detection, Supervision, and Safety for Technical Processes* (SAFEPROCESS'94 : IFAC Symposium, Helsinki University of Technology). International Federation of Automatic Control, 1994.

[26] M. Psarakis *et al.*, "Microprocessor software-based self-testing," *IEEE Design Test of Computers*, 2010.

[27] R. McLaughlin *et al.*, "Automated Debug of Speed Path Failures Using Functional Tests," in *27th IEEE VLSI Test Symposium*, 2009.

[28] K. Maragos *et al.*, "In-the-Field Mitigation of Process Variability for Improved FPGA Performance," *IEEE Transactions on Computers*, 2019.

[29] G. D. Natale *et al.*, *Cross-Layer Reliability of Computing Systems*. Jan. 2020.

[30] T. Hastie *et al.*, "The elements of statistical learning: Data mining, inference, and prediction," *Springer Science and Business Media*, 2009.

[31] S. Fralick, "Learning to recognize patterns without a teacher," *IEEE Transactions on Information Theory*, 1967.

[32] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, 1959.
[33] J. G. Carbonell *et al.*, "An Overview of Machine Learning," *Machine Learning*, 1983.
[34] A. Khan *et al.*, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, 2020.
[35] L. Alzubaidi *et al.*, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data*, Mar. 2021.
[36] J. T. Barron, "Continuously Differentiable Exponential Linear Units," *arXiv*, 2017.
[37] V. Borisov *et al.*, "Deep Neural Networks and Tabular Data: A Survey," *IEEE Transactions on Neural Networks and Learning Systems*, Dec. 2022.
[38] M. Long *et al.*, "Learning transferable features with deep adaptation networks," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.
[39] I. Goodfellow *et al.*, *Deep Learning*. MIT Press, 2016.
[40] O. Chapelle *et al.*, "Semi-supervised learning," *IEEE Transactions on Neural Networks*, 2009.
[41] H. Almousli *et al.*, "Semi Supervised Autoencoders: Better Focusing Model Capacity during Feature Extraction," in *Neural Information Processing*, M. Lee *et al.*, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.
[42] X. Zhu, "Semi-Supervised Learning Literature Survey," Computer Sciences, University of Wisconsin-Madison, Tech. Rep., 2005.
[43] G. Kostopoulos *et al.*, "Semi-Supervised Regression: A Recent Review," *Journal of Intelligent & Fuzzy Systems*, 2018, 2.
[44] I. Guyon *et al.*, "An Introduction to Variable and Feature Selection," *The Journal of Machine Learning Research*, Mar. 2003.
[45] I. T. Jolliffe *et al.*, "Principal component analysis: A review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, Apr. 2016.
[46] I. Guyon *et al.*, "Gene Selection for Cancer Classification Using Support Vector Machines," *Machine Learning*, Jan. 2002.
[47] L.Sunitha *et al.*, "Automatic Outlier Identification in Data Mining Using IQR in Real-Time Data," *International Journal of Advanced Research in Computer and Communication Engineering*, 2014.
[48] T. Chai *et al.*, "Root Mean Square Rrror (RMSE) or Mean Absolute Error (MAE)?– Arguments Against Avoiding RMSE in the Literature," *Geoscientific Model Development*, Jun. 2014.
[49] R. Williams *et al.*, "The Effect of Guardbands on Errors in Production Testing," in *Third European Test Conference (ETC)*, 1993.
[50] T. Kilian *et al.*, "An efficient high-volume production performance screening using on-chip ring oscillators," in *2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2023.
[51] Q. Lv *et al.*, "Enhanced-Random-Feature-Subspace-Based Ensemble CNN for the Imbalanced Hyperspectral Image Classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2021.
[52] Baosenguo, "1D-CNN: Kaggle-MoA 2nd Place Solution," *Kaggle*, Dec. 2020.
[53] L. Bottou *et al.*, "Optimization Methods for Large-Scale Machine Learning," *Society for Industrial and Applied Mathematics (SIAM) Review*, 2018.
[54] S. Geman *et al.*, "Neural Networks and the Bias/Variance Dilemma," *Neural Computation*, 1992.

Nicolò Bellarmino is a PhD Student in Computer and Control Engineering at Politecnico di Torino. He received his MS degree in Computer Engineering from Politecnico di Torino in 2021. His main research interest are Machine Learning, Data Analysis, and AI systems and their application to real-world cases. He worked in Machine Learning applied to device testing and reliability since 2020. He is part of IEEE-HKN.

Riccardo Cantoro received the MS degree and the PhD in computer engineering from Politecnico di Torino, Italy, in 2013 and 2017, respectively. He is currently a researcher with the Department of Computer Engineering of the same university. His research interests include software-based functional testing of SoCs and memories, and machine learning applied to test and diagnosis. He is a member of the IEEE.

Martin Huch received the Dipl.Ing. and Dr.Ing. degrees in electrical engineering from the Technical University of Darmstadt (TUD), Germany, in 1986 and 1991. After 5 years of digital circuit design with Bosch, Reutlingen he joined the TriCore design team of Siemens Corporation, Munich in 1997, which was later carved out to become part of Infineon. After some years of SOC design he transitioned to product engineering, where he "baby-sitted" all of Infineon's TriCore products from the very first samples until volume production. Focus topics are general analysis methodology, power integrity, performance validation.

Tobias Kilian received the B.Sc. and M.Sc. degree in electrical engineering and information technology from the Technical University of Munich (TUM), Munich, Germany, in 2017 and 2019, respectively. He is currently pursuing the Ph.D. degree as part of a collaborative project between Infineon Technologies A.G. and the Technical University of Munich. His research focus lies on performance monitors for automotive microcontrollers.

Ulf Schlichtmann (Senior Member, IEEE) received the Dipl-Ing and Dr-Ing degrees in electrical engineering and information technology from the Technical University of Munich (TUM), Munich, Germany, in 1990 and 1995, respectively. He is a professor and the head of the Chair of Electronic Design Automation, TUM. He joined TUM in 2003, following 10 years in industry. His current research interests include computer-aided design of electronic circuits and systems, with an emphasis on designing reliable and robust systems. Increasingly, he focuses on emerging technologies, such as lab-on-chip, and photonics.

Giovanni Squillero (Senior Member, IEEE) received a Ph.D. in Computer Engineering from Politecnico di Torino in 2002; his research mixed computational intelligence and machine learning, with industrial applications that range from electronic CAD to bio-informatics. Currently, Squillero is an associate professor of Computer Science at Politecnico di Torino, Department of Control and Computer Engineering; he is serving in the technical committee of the IEEE Computational Intelligence Society Games, and in the editorial board of Genetic Programming and Evolvable Machines.