Enabling Inter-Product Transfer Learning on MCU Performance Screening

Enabling Inter-Product Transfer Learning on MCU Performance Screening

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

Abstract—**In safety-critical applications, microcontrollers must meet strict quality and performance standards, including the maximum operating frequency ($F_{\max}$). Machine learning (ML) models can estimate $F_{\max}$ using data from on-chip ring oscillators (ROs), making them suitable for performance screening. However, when new products are introduced, existing ML models may no longer be suitable and require updating. Training a new model from scratch is challenging due to limited data availability. Acquiring $F_{\max}$ data is time-consuming and costly, resulting in a small labeled dataset. However, a large amount of data from legacy products may be available, along with existing ML models. In order to address the scarcity of labeled data, this paper proposes using deep learning feature extractors trained on specific MCU product data and fine-tuning them for new devices, in a Transfer Learning fashion. Experimental results show that these models can extract useful general features for performance prediction. As a result, they achieve better performance with significantly less labeled data compared to traditional shallow learning approaches.**

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

I. Introduction

The automotive and aerospace industries demand highly reliable devices, particularly microcontrollers (MCUs) used in safety-critical components. The performance screening of MCUs focuses on identifying devices that do not meet specified characteristics, such as the maximum operating frequency ($F_{\max}$) [1]. Traditional speed binning involves executing critical functional tests at increasing clock frequencies to identify underperforming devices. However, this approach is time-consuming, requires expensive Automatic Test Equipment (ATE), and only provides categorical binning results [1]–[5].

To address these challenges, ML regression models have been proposed to predict the $F_{\max}$ of MCUs based on correlated data [5]. ML techniques offer significant time savings compared to traditional speed binning methods [5]. Previous research has explored the use of on-chip ring oscillators, known as Speed Monitors (SMONs), as features to predict $F_{\max}$ values [3], [4], [6], [7]. These ML models establish the relationship between SMON values and the $F_{\max}$.

The accuracy of supervised ML models relies on high-quality and sufficient labeled data. While unlabeled data are readily available and inexpensive to obtain, acquiring $F_{\max}$ is time-consuming. When introducing a new product, there is often a scarcity or absence of data. Constructing an ML model from scratch under these circumstances becomes challenging, as it requires obtaining an adequate training set, which is very time-consuming. Additionally, using the very same model trained on previous-generation data may not be viable due to potential shifts in data distributions, resulting in significant prediction errors. However, data and models from previous-generation products remain accessible and can be utilized to build new ML models.

In this paper, we propose leveraging the knowledge acquired from a legacy MCU product *(P1)* to develop models for a different new product *(P2)* using Transfer Learning. Specifically, we employ 1-Dimensional Convolutional Neural Networks pre-trained on *P1* as Feature Extractors for *P2*. Our experiments demonstrate that this approach achieves a reasonable prediction error (2% normalized root-mean-square error, nRMSE) even with a limited number of samples (units). This accelerates the deployment of ML models in production and requires training only a shallow linear regression model to link the extracted features with the actual operating frequency. The performance achieved with this approach is comparable to classical shallow-learning algorithms, such as Ridge Regression with polynomial feature transformation, trained on thousands of samples.

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 (Section III-A) and an introduction to Deep Convolutional Networks and Transfer Learning (Sections III-B and III-C). In Section IV, the motivations for deploying transfer learning are given. In Section V, we detailed the proposed approach. Section VI presents the experimental evaluation. Finally, Section VII draws the conclusions.

II. Related Work

The usage of machine learning (ML) models to establish a relationship between structural and functional $F_{\max}$ was initially introduced in [3]. Subsequently, several approaches have been proposed for performance prediction [6]–[8].

The use of indirect measures to predict circuit characteristics, known as "alternate test", has been studied for analog circuits [9]–[12]. The idea is to learn a mapping between indirect measurements and circuit specifications. In this work, we leverage ML techniques to build this mapping, specifically

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

from Ring Oscillators (ROs) values (namely, Speed Monitors or SMONs) to $F_{\max}$.

In the field of MCU performance screening, researchers have worked on deriving ML models for $F_{\max}$ prediction [5], [7], [13], [14]. In [7], authors correlated the values of 27 SMONs obtained during wafer sort to functional $F_{\max}$. In [15], they found polynomial ridge regression (Poly Ridge) to be an effective ML model for MCU performance screening. In [13], Active Learning (AL) was employed to reduce the training set size by selecting informative samples for deriving the ML model. Additionally, outlier detection techniques were evaluated for identifying anomalous, noisy data and outliers [14].

In [16], Semi-Supervised Learning (SSL) was leveraged to develop Deep Feature Extractor models, showcasing how Neural Networks can acquire relevant knowledge even from unlabeled data. This work lays the foundation for applying such models in scenarios characterized by limited labeled data.

Transfer learning has emerged as a powerful technique in various domains [17]–[19]. The concept of pretraining and fine-tuning was introduced in [20], 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.

III. Background

A. Data Collection

The ML training process involves acquiring a suitable dataset, and thus, characterizing the MCUs. The measurements obtained from the on-chip ROs, known as SMONs, serve as the features for the ML models [7]. These SMONs' frequencies are accurately measured during production while the dies are still on the wafer, providing high-quality features in a stable, fast, and straightforward process. The SMONs measurement is part of the regular production test flow [15], as depicted in Figure 1.

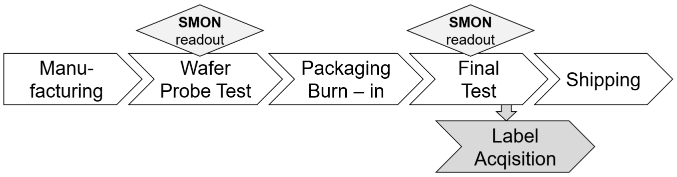

Fig. 1. Data collection steps through the manufacturing

On the other hand, labels acquisition is a separate and time-consuming process that is not included in the standard production test flow. Labeling involves individually measuring each MCU using functional test patterns [7]. This process requires mounting the MCU on a measurement board that simulates real-field applications. The MCU executes a specific functional pattern under worst-case voltage (V_{crit}) and temperature (T_{crit}), gradually increasing the frequency until a functional failure is observed [21]. The last working frequency $F_{\max}$ is then recorded. Multiple functional test patterns are typically employed, resulting in a multi-label dataset. The $F_{\max}$ values collected with different patterns can exhibit significant variations. The most critical pattern, denoted as $P_{\min}$, is the one with the lowest $F_{\max}$ value, which may differ among MCUs. Due to the substantial effort involved, the labeling process is conducted on a small subset of manufactured devices, leading to a scarcity of labeled data. This presents a major bottleneck in ML-based procedures [13], [15]. Consequently, techniques such as Outlier Detection [14], Active Learning [13], [15], and possibly Transfer Learning are employed to optimize the information extracted from the available data. In the end, the data acquisition process leads to multi-label numerical tabular data.

B. Transfer Learning

Transfer learning is an ML technique that involves using knowledge gained from solving one problem to improve the performance achieved on a different but related task. In the context of deep neural networks, transfer learning refers to leveraging pre-trained models on large datasets and transferring their learned knowledge to new tasks or domains. Deep neural networks are often trained on large-scale datasets that usually contain millions of samples. In computer vision applications, for example, it is quite common to start from a network pre-trained on very huge datasets with millions of labeled images. These models learn to extract hierarchical representations of data, enabling them to recognize various features and patterns. Instead of training a deep neural network from scratch on a new task or dataset, transfer learning allows initializing the network with the pre-trained weights, which serves as a valuable starting point. The first layers of the pre-trained neural network are then frozen to prevent their weights from being modified during the successive fine-tuning. The pre-trained layers serve as a feature extractor, capturing general patterns and representations from the original dataset. Due to the flexible nature of the neural networks, it is possible to add new custom layers on top of the pre-trained layers. These layers are task-specific and can be designed according to the requirements of the target task. After adding custom layers, it is possible to optionally unfreeze some pre-trained layers to allow them to be fine-tuned on the final target task. This step is beneficial when a substantial amount of data is given for the new task, allowing the model to adapt further to the specific domain. Transfer learning offers several benefits, including reducing the amount of required labeled data, speeding up training time, and improving generalization to new tasks. It is widely used across various domains, including computer vision, natural language processing, and audio processing.

C. Deep Convolutional Neural Networks

Deep Convolutional Neural Networks (CNNs) have emerged as a powerful machine learning model, especially in the field of computer vision. Nowadays, CNNs have achieved state-of-the-art results in a wide range of both supervised and unsupervised tasks such as image classification, object detection, and semantic segmentation or blind feature extraction. Supervised CNNs have been extensively used for tasks where labeled training data is available. The architecture of a typical supervised CNN consists of multiple layers, including convolutional, pooling, and fully connected layers. These networks are designed to automatically learn hierarchical representations of visual data directly from the raw input. The convolutional layers in a CNN apply a set of learnable filters to the input image. These filters are convolved with the input to produce feature maps, which are then passed through non-linear activation functions, such as rectified linear units (ReLU), to introduce non-linearity. Pooling layers are employed to downsample the feature maps and reduce spatial dimensions, aiding in translation invariance and computational efficiency. The fully connected layers at the end of the CNN receive the high-level features extracted by the previous layers and perform classification or regression tasks. These layers connect all neurons in one layer to all neurons in the next layer, allowing for complex relationships to be learned. CNNs can also be deployed in unsupervised tasks, to learn a useful representation from unlabeled data. A popular approach in unsupervised learning is the use of convolutional autoencoders. These models consist of two parts: an encoder network that maps the input data to a compressed representation, and a decoder network that reconstructs the input from the encoded representation. This process encourages the network to learn meaningful representations that capture important features and structures in the data. Unsupervised CNNs have been used for tasks like dimensionality reduction, feature extraction, and image denoising [22]. CNNs are mainly used in the field of images because of the ability of convolutional layers to catch spatial structure and patterns of data (such as correlation among near pixels in images). Convolutional kernels are designed to extract features by leveraging two important properties of input images: local connectivity and spatial locality [23]. The concept of local connectivity implies that each convolutional kernel operates on a small region of the input image during the convolution process. This locality property assumes that the pixels being convolved are closely related, and processing them together allows for the extraction of meaningful feature representations. For example, a single convolutional kernel can learn to capture edges, textures, shapes, gradients, and other relevant features. Interestingly, convolutional layers can also be applied to tabular data [16], [24]. In such cases, the tabular data can be transformed into images, enabling the straightforward application of CNNs. Alternatively, a one-dimensional convolution can be used to operate on the columns of the dataset, treating each sample with C dimensions as an image of size (1, C). CNNs have been effectively used in the field of MCU performance screening [16], catching non-linear relationships among input features.

IV. Transfer Learning: Reason Why

Extensive research has been conducted in recent years on predictive models for MCU performance screening [7], [13], [14]. Recently, the utilization of unlabeled data and Semi-Supervised Learning (SSL) techniques has emerged as a promising approach in this field, enabling the use of deep neural networks as feature extractors [16]. Once the knowledge of the task becomes mature based on the analysis of thousands of labeled devices of a specific MCU type, the predictive test can be applied to new products. However, applying the very same model, trained on legacy products, to new ones is infeasible as it leads to significant prediction errors, making ML-based performance prediction unreliable.

One potential solution is to build a new ML model from scratch, 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.

In scenarios where data scarcity is a concern, techniques such as Active Learning (AL) [13] and Outlier Detection [14] prove useful in creating an informative training set efficiently, thus facilitating the development of robust ML models. However, these approaches may not be sufficient when the number of available samples is extremely limited. Despite having a high-quality dataset, models trained with limited samples may exhibit biases and struggle to generalize well to unseen data.

In [16], the authors analyzed the time required to build a shallow learning supervised model from scratch. They found that obtaining a suitable training set for learning the ML model would require 63 days of continuous labeling (for about 3000 samples). 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 months. However, when deep learning feature extractor models are combined with SSL techniques, the time required can be significantly reduced to just a few days (2 to 10). These deep feature extractor models are trained on millions of easily obtainable unlabeled or pseudo-labeled data. Moreover, while traditional shallow learning models such as Support Vector Machines (SVM), Ridge Regression, and ElasticNet tend to outperform deep learning models in contexts with limited data, this advantage does not hold in big-data domains.

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 [15]. In contrast, deep learning models do not necessarily require this step or do not consider it as crucial. These ML algorithms have gained immense popularity due to their superior accuracy when applied to big data. They autonomously learn how to find the best data representation by manipulating input features.

However, when dealing with limited labeled data, the aforementioned DL models cannot be applied. This situation commonly arises when manufacturing a new product, necessitating precise strategies to address this problem. Instead of waiting to collect a sufficient number of labeled data or relying on large-scale mass production to obtain millions of unlabeled data, Transfer Learning can be employed. Typically, data and models from different or previous products are still available.

These 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.

The rationale behind using Transfer Learning becomes evident: it saves 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. The flexibility of deep learning models allows pre-training on a specific training set, often composed of a massive number of samples, followed by fine-tuning of the last layers on a related dataset where a similar task needs to be solved.

V. Proposed Approach

We aim to use Deep Neural Networks, pre-trained on millions of samples of a certain product *P1*, and use them as off-the-shelf feature extractors on a second product *P2*, demonstrating the advantages of Transfer Learning. We initially conducted two baseline experiments. First, we applied the very same model trained on *P1* directly on *P2*, with no fine-tuning. Next, we trained a shallow-learning algorithm (Polynomial Ridge Regression) only on samples from *P2*, without considering previous knowledge. These two baseline approaches are then compared with Transfer Learning techniques. The networks used as feature extractors are two Convolutional Neural Networks (the architecture is explained in detail in Section VI-B). The networks are pre-trained on unlabeled devices coming from production lines, available in millions of samples, by using Semi-Supervised Learning techniques, and the architecture has proven to be effective in the context of MCU performance screening [16].

The first network is a supervised CNN, trained on the dataset $D_1^{P1} = (X_{unl}^{P1}, \hat{y}^{P1})$, in which X_{unl}^{P1} are the SMON values of the product *P1* and $\hat{y}^{P1}$ are pseudo-labels predicted by using a shallow-learning model trained on labeled devices from *P1* (in a Semi-Supervised Fashion, [16]). The second network is an unsupervised Convolutional Auto-Encoder, trained on the dataset $D_2^{P1} = (X_{unl}^{P1}, X_{unl}^{P1})$, in which X_{unl}^{P1} are the SMONs values of product *P1*. This network is composed of two sub-parts: an Encoder, which projects the original feature space into a latent space, and a Decoder, which aims to reconstruct the original feature space from the latent one.

These models are used "as-it-is", in the sense that the layers are frozen, and no parameters are changed. Also, no hyperparameters tuning is needed, since they are not re-trained during the process but are available from legacy products. Alternatively, it is possible to re-train these deep models on the available labeled data from *P2*, to adapt the parameters of the convolutional kernels. However, if the representation learned by the network on *P1* is general enough, this step does not lead to an enormous improvement in the final prediction error evaluation. Also, a high number of samples could be required to fine-tune the intermediate layers. Once the deep feature extraction models are applied to the data from *P2*, a transformed dataset $D^{P2} = (\hat{X}^{P2}, y^{P2})$ is obtained, in which $\hat{X}^{P2}$ is the projection of X^{P2} in the latent space learned by the Neural Network. At this point, it is possible to train a simple Linear Regression model (in our case, a Ridge Regression) on D^{P2}, to learn how to map the transformed SMON values to the actual $F_{\max}$. In the test phase of these models on *P2*, the SMON values are first transformed with the deep feature extractor, and the Linear Ridge Regressor is then applied to compute the predicted operating frequency. The overall procedure is summarized in Fig. 2.

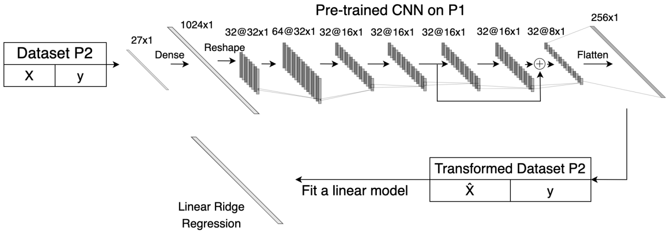

Fig. 2. Example of the proposed approach using the PL-CNN.

VI. Experimental Evaluation

A. Experimental Setup

The proposed methodology has been validated on a dataset composed of 1015 labeled *P2*-devices. The deep neural networks were trained by using 2986 labeled *P1*-devices, and 1,496,248 unlabeled samples from production lots. *P1* and *P2* devices come from the same product family, with the same SMONs equipped on board. Thus, for both products, we have 27 SMONs. The values of these SMONs are used as features by our ML models. Features are standardized by removing the mean and scaling to unit variance, as a pre-processing step. For *P1*, 10 labels are available, measuring the $F_{\max}$ for different functional test patterns. For *P2*, 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-A). 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.

The DL models were trained on *P1*, and the setup is described here: 85% of the training data set was used to train the models, that 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*, [25]). We used a Stochastic Gradient Descent (SGD) optimizer [25], [26] to tune the weights of each layer, with learning rate lr equal to 1×10^{-4} and momentum equal to 0.7 [25]. The lr decreases on loss plateau until 1×10^{-5}. The very same approach was used to eventually fine-tune the deep models. Each convolutional layer is preceded by a Batch Normalization layer. Results are presented in terms of normalized Root Mean Square Error (nRMSE), normalized Mean Absolute Error (nMAE), Learning Curves and Guardband G. RMSE and MAE are popular regression performance indexes [27], 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. To accommodate potential errors and uncertainties in statistical predictions, a risk-based guardband is necessary. 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 won't compromise the product's quality standards or reliability. Essentially, the error guardband acts as an additional layer of assurance during the production testing phase [16], [28]. 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 $f_{screen} + G$. The goal to achieve is to have G as small as possible, as it affects the production yield. We can compute G on a test set with true frequencies y, predicted frequencies $\hat{y}$ and errors $e = y - \hat{y}$ as:

$$G = \mu_e + k\sigma_e \tag{1}$$

μ_e and σ_e are the mean and the standard deviation of the error distribution and k is a parameter that permits to choose the defects' level in ppm. $k = 5.2$ is an approximation for 0.1 ppm, but we used a more stringent value ($k = 6$) 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 × 16, 32GB of RAM, and an Nvidia® 2080 TI GPU.

B. Deep-Learning Models

We used the following DL models, already trained on *P1* (as described in the Section V):

1) 1D-CNN with skip connection (namely CNN with Pseudo-Labeling, PL-CNN): CNN with a fully connected layer as the first layer, with CELU (Continuously differentiable Exponential Linear Units) activation function [29]. This first layer project the features into a higher dimensional space by mean of non-linear combinations (see Fig. 2), reordering features (soft-ordering) and enabling 1D-convolutional layers to extract relevant features-interaction [16], [30] The output of the first layer is then reshaped into image-like samples of dimension $(H, 1, C)$ (height H, length 1, and channels C). We used batch normalization and dropout layers between the convolutional layers.
2) Convolutional AE (namely AE-CNN): AE with soft-ordering, 1D convolutional encoder, and deconvolutional decoder. The encoder performs feature space augmentation. The output of the encoder is flattened before going into the final supervised model.

C. Results

As show in Table I, applying a model trained on *P1* directly on *P2* 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 P2 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 I). 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. 3). 2% value of nRMSE has been identified as a prediction error that leads to an acceptable yield in the process, in terms of guardband. Experiments showed that the feature extractor DL models, trained on *P1*, can extract relevant information from *P2*. This is true even without fine-tuning (FT, in the plots) the intermediate convolutional layers, meaning that the feature manipulation performed by the neural networks is general enough, and can be successfully applied to the different product. Fine-tuning the intermediate layers model does not lead to improvement, and this may be due to the low amount of training data from *P2*, not sufficient to properly update the entire networks (that have a number of parameters on ten-thousands of magnitude). 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. By using the PL-CNN, building and deploying ML-predictive model for the operating frequency of a new MCU requires just $30 \text{ min} \cdot 16 \text{ samples} = 480 \text{ min } = 8$ hours. Also, no additional feature engineering phase is required, due to the ability of the NNs to extract features autonomously.

VII. Conclusions

We presented a Transfer Learning approach for optimizing the ML-based MCU performance screening. We made use

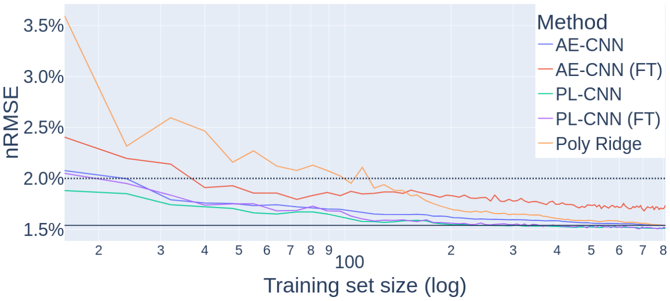

Fig. 3. Learning curves for different models. The upper dotted horizontal line is 2% of nRMSE. The lower horizontal line is 1.54% nRMSE, the final error obtained by the Polynomial Ridge. The x-axe is the number of training samples (log scale) while the y-axe is the nRMSE computed on the test set. Deep Feature extractors are able to obtain lower prediction error with a smaller training set.

TABLE I
RESULTS ON P2-PRODUCT (AVERAGE ON 5 TRAINING-TEST SPLITS)

Model	nRMSE	nMAE	G	Samples to 2% nRMSE
Poly Ridge (P1)	33.93%	33.63%	77.38%	–
Poly Ridge (P2)	1.54%	1.16%	11.19%	121
PL-CNN	**1.51%**	**1.13%**	**11.01%**	**16**
PL-CNN (FT)	1.52%	1.14%	11.05%	24
AE-CNN	1.54%	1.15%	11.21%	24
AE-CNN (FT)	1.74%	1.33%	12.68%	40

of deep neural networks as feature extractors, pre-trained on a legacy product *P1* and thus available when a new product comes. This enables the use of Transfer Learning on a new product *P2*. Transfer Learning permitted us both to reduce the prediction error of our models (reaching 1.51% nRMSE) and decrease the number of labeled samples needed to train the supervised models, reducing the time needed to build an effective ML training set. With this approach, only hours of labeling are required (not days/months). The feature extraction step presented in this paper can be used in other scenarios in which pre-trained models are available, to re-use previous knowledge on different products. Future works aim at optimizing the deep learning models, by tuning the NNs architecture to achieve further improvement in the learning process and an additional reduction in the samples needed to build the supervised models.

REFERENCES

[1] J. Zeng *et al.*, "On correlating structural tests with functional tests for speed binning of high performance design," in *Fifth International Workshop on Microprocessor Test and Verification (MTV'04)*, 2004.

[2] B. D. Cory *et al.*, "Speed binning with path delay test in 150-nm technology," *IEEE Design Test of Computers*, 2003.

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

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

[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] 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.

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

[8] 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.

[9] H. Ayari *et al.*, "Making predictive analog/rf alternate test strategy independent of training set size," in *2012 IEEE International Test Conference (ITC)*, 2012.

[10] P. Variyam *et al.*, "Prediction of analog performance parameters using fast transient testing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2002.

[11] H.-G. Stratigopoulos *et al.*, "Error moderation in low-cost machine-learning-based analog/rf testing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2008.

[12] J. Brockman *et al.*, "Predictive subset testing: Optimizing ic parametric performance testing for quality, cost, and yield," *IEEE Transactions on Semiconductor Manufacturing*, 1989.

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

[14] 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.

[15] 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.

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

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

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

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

[20] 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.

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

[22] P. Vincent *et al.*, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, 2010.

[23] I. Goodfellow *et al.*, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[24] V. Borisov *et al.*, "Deep neural networks and tabular data: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, Dec. 2022.

[25] L. Bottou *et al.*, "Optimization Methods for Large-Scale Machine Learning," *SIAM Review*, 2018.

[26] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[27] T. Chai *et al.*, "Root mean square error (RMSE) or mean absolute error (MAE)?– Arguments against avoiding RMSE in the literature," *Geoscientific Model Development*, Jun. 2014.

[28] R. Williams *et al.*, "The effect of guardbands on errors in production testing," in *Proceedings ETC 93 Third European Test Conference*, 1993.

[29] J. T. Barron, "Continuously Differentiable Exponential Linear Units," *arXiv*, 2017.

[30] Baosenguo, "1D-CNN: Kaggle-MoA 2nd Place Solution," *Kaggle*, Dec. 2020.