Semi-Supervised Deep Learning for Microcontroller Performance Screening

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

***Abstract*—In safety-critical applications, microcontrollers must satisfy strict quality constraints and performances in terms of $F_{\max}$ (the maximum operating frequency). Data extracted from on-chip ring oscillators (ROs) can model the $F_{\max}$ of integrated circuits using machine learning models. Those models are suitable for the performance screening process. Acquiring data from the ROs is a fast process that leads to many unlabeled data. Contrarily, the labeling phase (i.e., acquiring $F_{\max}$) is a time-consuming and costly task, that leads to a small set of labeled data. This paper presents deep-learning-based methodologies to cope with the low number of labeled data in microcontroller performance screening. We propose a method that takes advantage of the high number of unlabeled samples in a semi-supervised learning fashion. We derive deep feature extractor models that project data into higher dimensional spaces and use the data feature embedding to face the performance prediction problem with simple linear regression. Experiments showed that the proposed models outperformed state-of-the-art methodologies in terms of prediction error and permitted us to use a significantly smaller number of devices to be characterized, thus reducing the time needed to build ML models by a factor of six with respect to baseline approaches.**

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

I. Introduction

Automotive and aerospace industries require high dependability in electronic devices, especially for microcontrollers (MCUs) used in safety-critical components. MCU performance screening aims to detect underperforming devices that do not fully meet the characteristics described in the datasheet in terms of maximum operating frequency $F_{\max}$. The $F_{\max}$ of a device is precisely determined in different worst-case conditions. In the traditional speed-binning, critical functional tests or benchmarks are executed on the devices at increasing clock frequency exciting the critical paths until a failure happens. These functional tests usually occupy a significant portion of the tester memory. Also, this approach requires an automatic test equipment (ATE) that can operate at the targeted frequency, which is usually extremely expensive [1]. Both the long test time and the high requirement of ATEs increase the cost of speed binning. A possible approach is to use machine learning (ML) regression models trained on data that can be correlated with the device's $F_{\max}$. This technique is not intended to replace structural testing, yet it is a much more informative process compared to a simple speed binning, where devices are merely sorted into categorical bins (e.g., "fast" and "slow") [1]–[6]. The gain in using ML techniques in speed-binning are enormous in terms of test-time saving [5].

Previous works have proposed the usage of on-chip ring oscillators - the so-called Speed Monitors (SMONs)- as features to predict $F_{\max}$ values derived from the execution of functional patterns on single devices [3], [4], [7], [8]. The obtained ML regression models will be able to relate the SMONs value with the continuous-value maximum frequency (or speed) of the devices.

The accuracy of supervised ML models depends on the quality and the quantity of labeled data. In this context, unlabeled data are relatively inexpensive to acquire. Instead, the process of acquiring $F_{\max}$ is costly in terms of time required, and thus the amount of available data is limited. Obtaining a proper training set for ML models of the size of thousand of samples could require several months.

In this paper, we focus on optimizing the training procedure of ML models by reducing as much as possible the number of labeled samples needed. We made use of deep learning (DL) to extract relevant features from a set of unlabeled data from production lines as a pre-training step for the successive supervised learning algorithms, in semi-supervised learning (SSL) fashion. The feature embedding created by DL models is then used to train a simple Linear Regression model, with $F_{\max}$ as a target. Experiments showed that this approach can lead to an acceptable prediction error (2% of normalized root mean square error, nRMSE) even with a fraction of labeled data (tens of samples). The resulting performances are comparable to classical shallow-learning algorithms (Ridge Regression with polynomial feature transformation) trained with thousands of samples.

The rest of the paper is organized as follows: Section II presents related works on the topic. Section III describes theory and concept useful for understanding successive experiments; in particular, Section III-A describes the data collection process to obtain the dataset for ML algorithms, while Section III-B introduces the concepts of DL and SSL. In Section IV, the motivations for using deep learning models over classical machine learning models are given. In Section V, details on the proposed approach are given. Section VI presents the experimental evaluation. Finally, Section VII draws the conclusions.

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

II. Related Work

Several approaches to performance prediction have been proposed in the past [6]–[9]. The use of ML models to relate structural and functional $F_{\max}$ was first introduced in [3].

Using indirect measures to predict a circuit characteristic is called 'alternate test' in literature, and has been widely studied for analog circuits [10]–[13]. The core idea is to learn a mapping between indirect measurements and some circuit specifications, and to use only the indirect low-cost measurements to predict device specifications during production testing.

The authors of [5], [8], [14], [15] worked on deriving an ML model for $F_{\max}$ prediction for MCU performance screening. In [8], they correlated the values of 27 speed monitors coming from wafer sort to functional $F_{\max}$ measured on more than 4,000 packaged devices extracted from 26 corner-lot wafers. In [14], authors moved a first step towards the reduction of the training set size in ML performance screening, by using three Active Learning (AL) metrics to select the most informative samples from which derive an ML model. In [15], they evaluated the effectiveness of several outlier detection techniques in identifying anomalous, noisy data, and outliers; as a result, they were able to increase the training set size by recovering incomplete samples, thus reducing the number of samples to be labeled for the training procedure.

Working with unlabeled data is a well-known topic in the ML community. SSL, transfer learning, and meta-learning are concepts that reached relevant importance in recent years [16]. The heuristic approach of self-training (also known as self-learning or self-labeling) is historically the oldest approach to SSL, with examples of applications starting in the 1960s [17], [18]. Nowadays, deep neural networks play a dominant role in many research areas. Classic SSL frameworks can be adapted to support DL settings. The research community proposed a considerable amount of Deep SSL methods [16]. There are many learning paradigms related to SSL that make use of an extra data source to boost learning performance. Transfer learning [19]–[21] aims to apply knowledge from one or more source domains to a target domain in order to improve performance on the target task. Meta-learning [22]–[24] aims to learn new skills or adapt to new tasks rapidly with previous knowledge and a few training examples.

III. Background

A. Data Collection

ML training process requires the acquisition of a proper dataset and, thus, the MCU characterization. In the ML context, the SMONs' measurements are the features. The SMONs are on-chip ring oscillators (ROs), and their frequencies are measured during the production, when the dies are still on the wafer, with high accuracy in a stable, fast, and easy process. Hence, the obtained features (RO-frequencies) have high quality. The SMON measurement is part of the regular production test flow, as shown in Fig. 1. Thus, each produced MCU is, by default, an unlabeled device -with only the SMON measurements.

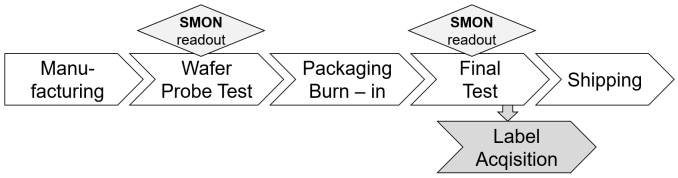

Fig. 1. Data collection steps trough the manufacturing

In contrast, label acquisition is a separate process that is not part of the usual production test flow. The labeling process is a time-consuming procedure performed mostly manually, in which each MCU is measured individually with functional test patterns [8]. The labeling process is done by mechanically mounting each MCU on a measurement board that mimics the in-field applications. Then, the MCU starts executing a certain functional pattern (e.g., a test program or a general purpose routine) with a low frequency, and the frequency is slowly stepped-up until observing a functional failure [25] (e.g., a crash in the application, an erroneous response, etc.); the last working frequency $F_{\max}$ is then stored. The procedure is typically repeated using various functional test patterns, thus leading to a multi-label dataset. Therefore, due to the high effort, the labeling process is performed on a small subset of the manufactured devices. For each MCU, the $F_{\max}$ values collected with the various patterns can have a considerable spread. The most critical pattern is the one with the lowest $F_{\max}$ value, and this is not necessarily the same for all MCUs. We call the critical pattern $P_{\min}$.

While the measurement of the features is highly accurate, the measurement of the label is more inaccurate: it is performed under worst-case voltage (V_{crit}) and temperature (T_{crit}) conditions at the limits of the specified operating window. Even minor deviations in individual parameters, mechanical vibrations or statistical noise may change the result.

B. Deep and Semi-Supervised Learning

SSL has emerged as a new research direction in ML. In classical SSL scenario, we dispose of two sets of data: the (few) labeled samples with both features $X_l = (x_1, \ldots, x_l)$ and labels Y_l, and a large number of unlabeled samples $X_u = (x_{l+1}, \ldots, x_{l+u})$. SSL falls between unsupervised learning (no labeled training data) and supervised learning (only labeled training data). SSL is relevant in the context in which it is expensive to produce labeled data, while the amount of unlabeled data is huge [26].

Different approaches to SSL have been developed: the simplest one is the *pseudo-labeling*, which consists of three steps: first, training a learning algorithm on a small subset of the labeled data; then, applying the just-trained model to the unlabeled samples, obtaining an approximate label (or prediction); finally, training the model with the whole labeled dataset (with actual and pseudo-labeled data).

A more robust method is the Self-Training, one of the earliest approaches in SSL [18], [26]. It is an iterative pseudo-labeling algorithm. At every iteration, only the most promising

pseudo-labeled data X_{ul} (promising on the basis of the probability of being correct, or confidence intervals) are selected. Then, it trains a new supervised classifier over $X_l \cup X_{ul}$ by considering these pseudo-labeled data as additional labeled points [27]. This procedure can only be used when the underling model is able to compute a confidence interval of the prediction (and thus, can be applied only for certain classification models).

DL models can be eventually integrated into SSL framework: Auto-encoders (AE) can be used in Semi-Supervised contexts as blind (or task-unaware) feature extractors [28], to learn potentially useful features for a subsequent supervised task. Over-complete AE project data into an higher dimensional space. Deep AEs are mainly of two types: fully connected or convolutional. Convolutional Neural Networks (CNNs) are used in all the state-of-the-art deep models in computer vision. Convolutional layers are effective in catching the spatial structure of data and spatially correlated patterns (such as among near pixels in images).

Convolutional kernels are feature extractors that exploit two properties of the input images: local connectivity and spatial locality [29]. The first means that each convolutional kernel is applied to a small region of the input image when performing the convolution. The spatial locality property means that the pixels where the convolutional kernel is applied should be highly correlated, and usually processing them jointly makes it possible to extract a meaningful feature representations. One convolutional kernel, for instance, can learn to extract edges, textures, forms, gradients, and other relevant features. Convolutional layers can be used also in the case of tabular data [30]: in this case, we can convert tabular data into images (and use straight-forward CNN) or use a 1-dimensional convolution that act on the columns of the dataset (considering each sample of C dimension as an image of size $(1, C)$).

IV. Deep learning: Reason Why

In the context of scarcity of data, AL [14] and Outlier Detection [15] techniques are definitely useful to create an informative training set in a fast and efficient manner, for building robust ML models.

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.

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. But DL models arise a huge amount of interest thanks to their supremacy in terms of accuracy when applied to big data. 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) and then fine-tune it on a related dataset, in which we want to solve a similar task. 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 [29], and more abstract representations computed in terms of less abstract ones, moving from input to output layers. Moreover, they are able to extract features in an autonomous way, rather than relying on a-priori hand-crafted features (as in shallow learning), and the architecture of the networks can be tuned without any limit.

These characteristics enable a pre-trained DL model to be employed as a feature extractor, by using only a portion of the architecture (e.g., the first n layers) to extract a relevant feature embedding, using the models as transformers or encoders [29] (as in the auto-encoders). For all these reasons, one can naturally think about switching to DL models, if the available data permit this. And this may happen even with a small amount of labeled data, if appropriate semi-supervised techniques are used (and if we have a great number of unlabeled data).

V. Deep Learning Approach

The approach followed in this work aims at extracting a feature representation of the data by means of DL models, used as transformers. The approach consists of four steps (see Fig. 2):

1) We train the models to perform a certain (alternative) task T_i using the unlabeled data.
2) We fine-tune the models using the labeled data, to adapt them to the actual labeled data distribution: starting from the solution (i.e., the layers' weights) obtained by the training on the unlabeled data, we run again the training optimization on the labeled data, in a warm-start fashion.
3) We drop the last layers of the models (i.e., the ones the most specialized and able to solve 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 (hopefully) solving the original performance prediction task is easier.
4) We train a linear Ridge Regression model using the features extracted in the previous step as inputs. Such a model will combine them to solve the performance prediction task by means of an L2 regularized loss (mean squared error).

This approach is something similar to *Transfer Learning* and *Meta Learning*: since the two domains are similar, we aim to transfer the knowledge (i.e., a feature transformation) acquired by the DL models fed with the unlabeled data to the labeled ones. In step 1, we used different alternate tasks T_i, described in the following Subsections.

A. Pseudo-labeling

We first build a supervised ML model with the 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 a training set $X_u = \{(x_{u_1}, \tilde{y}_{u_1}), (x_{u_2}, \tilde{y}_{u_2}), \ldots, (x_{u_n}, \tilde{y}_{u_n})\}$. Each pseudo-label $\tilde{y}_{u_i}$ may not be accurate, but it is directly linked to the final task we aim to solve (i.e., performance prediction). A

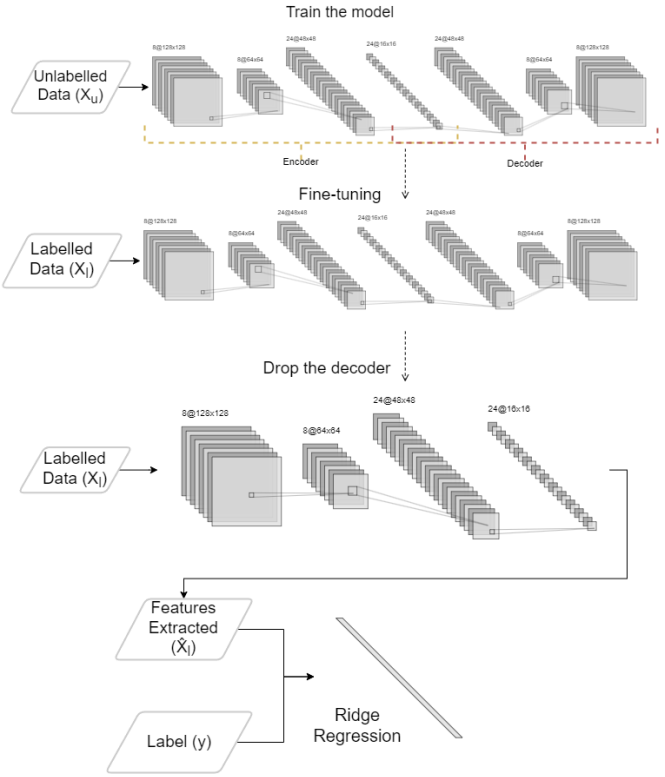

Fig. 2. Example of the proposed approach using a convolutional AE.

supervised DL model is then trained on the training set X_u.The model is then fine-tuned on the actual labeled training set. Since the model is not "blind" to the problem we aim to solve, we expect that the feature extracted will be relevant for the performance prediction task.

B. Auto-Encoder

The task is based on trying to reconstruct the initial input. We define the training set as $X_u = \{(x_{u_1}, x_{u_1}), (x_{u_2}, x_{u_2}), \ldots, (x_{u_n}, x_{u_n})\}$, so the input and the output of the models are the same. The model can be divided into two parts: an encoder (which projects the data into an alternate space) and a decoder (which learns how to recover the original data from the alternate space). This task is totally unsupervised because we are using only the features x of the unlabeled set. At the end of the process, we drop the decoder and use only the encoder as features extractor. This task is "blind" and not aware of the actual problem we aim to solve.

C. Denoising Auto-Encoder

Similar to the previous task, but here the inputs of the model are a noisy version of the data (with a superimposed Gaussian noise). The training set is thus $X_u = \{(\tilde{x}_{u_1}, x_{u_1}), (\tilde{x}_{u_2}, x_{u_2}), \ldots, (\tilde{x}_{u_n}, x_{u_n})\}$, 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.

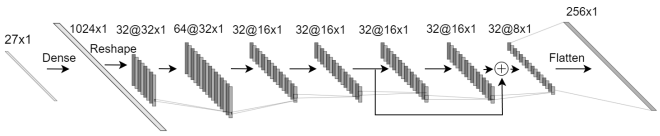

Fig. 3. Example of 1D-CNN with soft-ordering and skip-connection that act as features extractor. The original input space (of dimension 27) is mapped to a higher dimensional output space (256)

VI. Experimental Evaluation

A. Experimental Setup

The proposed methodology has been validated on a dataset composed of 4,039 devices coming from 26 different wafers, with 27 features (SMONs values) and 10 labels (functional test patterns), plus the final artificial target $P_{\min}$ (the minimum among the 10 patterns, Section III-A). In order to deal with outliers data and missing values, we applied an Outlier Detection procedure (IQR and z-score) and Imputation (Iterative Imputer with Random Forest) as pre-processing step [15]. The final training set is composed of 2,986 samples. The test set used for the evaluation of all the models is composed of 602 samples. We used 1,496,248 unlabeled device samples from 280 production lots (5,498 wafers). These samples were divided into training and test sets (75%-25%) (the test set was used for sanity check)

In the denoising task, each feature was corrupted with Gaussian noise of standard deviation equal to $\frac{1}{10}$ of the original feature standard deviation. For the pseudo-labeling task, we used a Polynomial Ridge Regression (Poly Ridge). It uses a polynomial transformation of the input features. It is considered the baseline method ([14], [15]).

At each step, the DL models are trained on 85% of the training data set and validated on the remaining 15%. The training procedure run 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*, [31]). We used a Stochastic Gradient Descent (SGD) optimizer [31], [32] to tune the weights of each layer, with learning rate lr equal to 1×10^{-4} and momentum equal to 0.7 [31]. The lr decreases on loss plateau until 1×10^{-5}. 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 [33], but in this context we normalized them 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 with respect to the mean frequency of the samples. 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 (we used a logarithmic scale) and on the y-axis the prediction error made by the model on the test set. 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 Poly Ridge, fine-tune the deep models, and train the final linear Ridge Regression. Then, the nRMSE, nMAE and G were computed on the test set (defined above). Since we deal with statistical prediction, they may be subject to error. For this reason, we need the adoption of a risk-based guardband that surrounds the specification limit, derived from the uncertainty in the predictions. G [34] is defined as how much increase the number of minimum product specifications to ensure that, even with prediction error, the product will meet the minimum specifications. In other words, the guardband is the amount of Hz by which the actual performance screening threshold frequency must be increased to ensure that no more than a specific ppm (part per million) of false-positive results occur in production. Therefore, the performance screening threshold frequency is the addition of the calculated guardband and the operating frequency specified in the data sheet. 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. 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 implemented the following DL models:

1) A Soft-Ordering 1D-CNN with skip connection (namely CNN with Pseudo-Labeling, CNN-PL): CNN with a fully connected layer as the first layer, with CELU (Continuously Differentiable Exponential Linear Units) activation function [35]. This first layer project the features into a higher dimensional space by mean of non-linear combinations (see Fig. 3). Since tabular dataset are not spatially correlated (as happen 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 by means of non-linear combinations, we may extract some relevant interactions between the original feature, and hopefully, a 1D-convolutional layer could catch those. This is the reason why we used a fully connected layer as the first layer. 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). Each of these signals 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. We used batch normalization and dropout layers between the convolutional layers.
2) Fully-connected AE (FC-AE): 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 (DFC-AE): as above, but trained on a corrupted version of the data.
4) Convolutional AE (CNN-AE): 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. 2).
5) Denoising Convolutional AE (DCNN-AE): as above, but trained on a corrupted version of the data.

C. Results

Experiments showed that almost all the DL models can extract relevant information from the unlabeled data, with reached prediction errors aligned to baseline Polynomial Ridge.

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), 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. 4). This means that the neural network has effectively found useful features for the underline regression problem, and it is able to well generalize on 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 (≈ 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. 4) with just tens of data (89 for CNN-PL, FC-AE and DFC-AE, 119 for CNN-AE, and 149 for DCNN-AE). The reason why the CNN-PL seems to be the best approach may be due to the awareness of the performance prediction task. Consequently, the features

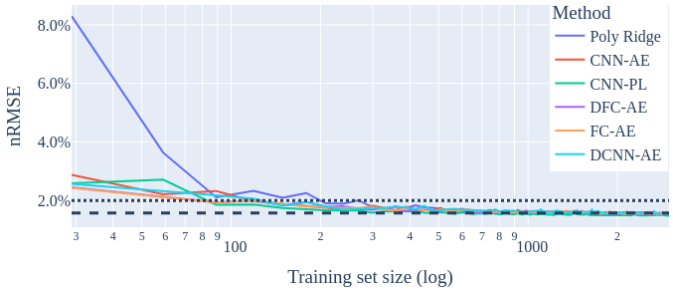

Fig. 4. Learning curves for different Deep Learning methods. The upper dotted line is the 2% of nRMSE. The lower line is the Poly Ridge prediction error (1.57% nRMSE). The x-axes is the number of training samples (log scale) while the y-axes is the nRMSE in percentage computed on the test set.

TABLE I
RESULTS ON THE TEST SET WITH DIFFERENT DEEP LEARNING ARCHITECTURES

Model	nRMSE	nMAE	G	Samples to 1.57% nRMSE
Poly Ridge	1.57%	1.17%	10.04%	2,986
CNN-PL	**1.48%**	**1.09%**	**9.43%**	**477**
FC-AE	1.56%	1.15%	9.96%	1,702
CNN-AE	1.52%	1.13%	9.70%	1,971
DFC-AE	1.55%	1.14%	9.95%	1,702
DCNN-AE	1.54%	1.14%	9.85%	1,822

extracted using this method are directly finalized to solve the actual problem, while the other methods perform a more general feature manipulation.

VII. CONCLUSIONS

We presented an approach based on semi-supervised learning for optimizing the MCU performance screening with ML techniques. SSL permitted us both to reduce the prediction error of our models (reaching 1.48 % of nRMSE) and decrease the number of labeled samples needed to build prediction models by a factor of six.

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 (9.43%), thus increasing the process yield, since the number of good devices incorrectly discarded would be reduced.

The developed SSL framework is absolutely general and can be deployed in almost every scenario with a huge amount of unlabelled data, such as MCU performance screening or alternate tests.

REFERENCES

[1] J. Zeng *et al.*, "On correlating structural tests with functional tests for speed binning of high performance design," in *2004 International Conference on Test*, 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 ITC*, 2009.

[4] J. Chen *et al.*, "Selecting the most relevant structural Fmax for system Fmax correlation," in *2010 VTS*, 2010.

[5] S. Mu *et al.*, "Statistical Framework and Built-In Self-Speed-Binning System for Speed Binning Using On-Chip Ring Oscillators," *IEEE VLSI*, 2016.

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

[7] M. Sadi *et al.*, "SoC Speed Binning Using Machine Learning and On-Chip Slack Sensors," *IEEE TCAD*, 2017.

[8] R. Cantoro *et al.*, "Machine Learning based Performance Prediction of Microcontrollers using Speed Monitors," in *2020 ITC*, 2020.

[9] 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.

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

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

[12] 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.

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

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

[15] 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.

[16] X. Yang *et al.*, *A survey on deep semi-supervised learning*, 2021.

[17] H. Scudder, "Probability of error of some adaptive pattern-recognition machines," *IEEE Transactions on Information Theory*, 1965.

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

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

[20] C. Tan *et al.*, *A survey on deep transfer learning*, 2018.

[21] F. Zhuang *et al.*, *A comprehensive survey on transfer learning*, 2019.

[22] T. Hospedales *et al.*, *Meta-learning in neural networks: A survey*, 2020.

[23] J. Vanschoren, *Meta-learning: A survey*, 2018.

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

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

[26] O. Chapelle *et al.*, "Semi-supervised learning (chapelle, o. et al., eds.; 2006) [book reviews]," *IEEE Transactions on Neural Networks*, 2009.

[27] M.-R. Amini *et al.*, *Self-training: A survey*, 2022.

[28] 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.

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

[30] V. Borisov *et al.*, *Deep neural networks and tabular data: A survey*, 2021.

[31] L. Bottou *et al.*, *Optimization methods for large-scale machine learning*, 2016.

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

[33] 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.

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

[35] J. T. Barron, *Continuously differentiable exponential linear units*, 2017.