Investigating the Resilience Source of Classification Systems for Approximate Computing Techniques

Mario Barbareschi, *Associate Member, IEEE,* and Salvatore Barone, *Member, IEEE,*

Abstract—During the last decade, classification systems (CSs) received significant research attention, with new learning algorithms achieving high accuracy in various applications. However, their resource-intensive nature, in terms of hardware and computation time, poses new design challenges. CSs exhibit inherent error resilience, due to redundancy of training sets, and self-healing properties, making them suitable for Approximate Computing (AxC). AxC enables efficient computation by using reduced precision or approximate values, leading to energy, time, and silicon area savings. Exploiting AxC involves estimating the introduced error for each approximate variant found during a Design-Space Exploration (DSE). This estimation has to be both rapid and meaningful, considering a substantial number of test samples, which are utterly conflicting demands. In this paper, we investigate on sources of error resiliency of CSs, and we propose a technique to haste the DSE that reduces the computational time for error estimation by systematically reducing the test set. In particular, we cherry-pick samples that are likely to be more sensitive to approximation and perform accuracy-loss estimation just by exploiting such a sample subset. In order to demonstrate its efficacy, we integrate our technique into two different approaches for generating approximate CSs, showing an average speed-up up to ≈18.

Index Terms—Approximate Computing, Design Space Exploration, Decision Trees, Deep Neural Networks

1 Introduction

Classification Systems (CSs) embrace different algorithms that aim to classify input data into predefined categories or classes. Most of them are based on machine learning techniques that exploit a set of labeled samples, namely the dataset, to generate a data model that is actually employed to classify other samples. These algorithms are widely adopted in a variety of applications, including speech recognition, image recognition, decision support, traffic packet inspection, natural language processing, and so forth.

During the last decade, with the advent of the Big Data era, CSs got a massive attention from the research community, shifting the focus mainly on classification performance, both in terms of accuracy and computational overhead. Indeed, while new learning algorithms can generate models that achieve high accuracy in many applications, their usage can be expensive, both in required hardware resources and execution time, especially when dealing with large models or when real-time processing is required.

For these systems, Approximate Computing (AxC) can be successfully adopted: it leverages a set of techniques that aim to balance the trade-off between result accuracy and efficiency of a computing system. The basic idea is to use techniques that allow computations to be performed with reduced precision or with approximate values, rather than exact values, to save energy, time, and, for hardware circuits, area. When approximate, some computing systems continue to behave almost correctly regarding the quality of the results of their calculations, exhibiting some resilience to errors. This is the case of CSs, which are inherently resilient to error, since the training set and data model exhibit a high degree of redundancy. The interesting results of Approximate Classification Systems (ACSs) implemented as hardware accelerator are [1], [2], in which authors successfully implemented approximate variants of Decision Tree based Mutiple Classifier Systems (DT MCSs), and [3], [4] in which approximate multipliers were adopted to implement artificial Neural Networks (NNs).

Generally speaking, to explore possible AxC variants of a given computing system, it is necessary to estimate introduced error through metric functions. In particular, for ACSs, a Design-Space Exploration (DSE) strategy could consider classification accuracy [5]. We can easily detect two main challenges related to classification accuracy estimation: i) test set relevance w.r.t. application and cardinality; ii) computational convenience of classification accuracy calculation.

As for the former, it is necessary to have a significant amount of test samples in order to effectively estimate accuracy of approximate variants of a classifier. As for the latter, since a new measure of accuracy is needed each time an approximate variant is found during the DSE, the estimation should take as little time as possible to get done. As one can see, such observations conflict with one another: the larger the test set, the more effective the accuracy estimation, but the longer execution time. Conversely, the smaller the test set, the less effective the accuracy estimation, but the shorter execution time.

In this paper, we first investigate the sources of error

Mario Barbareschi and Salvatore Barone are with the Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Naples, Italy.
E-mail: [firstname.lastname]@unina.it
Authors are listed in alphabetical order.
Manuscript received April 19, 2005; revised August 26, 2015.

resiliency in CSs that are exploited by the AxC, then we profitably use such sources to propose a technique to reduce time to calculate the accuracy loss of ACS variants by significantly reducing the test set, selecting test samples that are likely to be more sensitive to the approximation. By doing so, we can estimate the accuracy of the approximate system more efficiently and accelerate the DSE process. To the best of our knowledge, this is the first paper providing a general approach to haste accuracy loss estimation when AxC is involved during a DSE. We integrate our technique into two different methodologies to generate approximate DT MCSs and NNs CSs, showing its efficacy through the experimental results. In particular, we show a speed-up to complete a DSE up to approximately 18. The remainder of this paper is organized as follows. Section 2 provides an overview of related work on ACSs and DSEtechniques. Section 3 shows how AxC can be modeled as perturbation source and provides some preliminary results about our proposed method. Section 4 presents experimental result on well-known and publicly available datasets and a thorough analysis that highlighs benefits of our method. Finally, Section 5 draws the conclusion of the paper and discusses future research directions.

2 Related works and state-of-the-art

Imprecise calculations can be selectively exploited to enhance computing system performance, according to the scientific literature, defining the AxC paradigm [6]. Indeed, due to the redundancy of inner calculations, some applications are characterized by an inherent resiliency to errors [5]. Basically, by relaxing the functional requirements of a computing system, AxC enables the trade-off of output accuracy for performance, but, unfortunately, exploiting it to its full potential requires addressing several challenges. Identifying portions of the application that are amenable to approximation as well as suitable approximation techniques, just to mention a few, are not trivial tasks since they require in-depth knowledge of the target application. Furthermore, assessing the error due to the approximation is mandatory to ensure that quality constraints are met, and only those approximate configurations delivering the best compromises between error and savings have to be selected [7], [8]. Despite these challenges, the AxC has already been successfully exploited in a vast plethora of application fields, both hardware and software. Hardware circuit design [9], [10], [11] and signal and image processing applications [12], [13], [14] are just a few examples.

Besides those mentioned, CSs are one of the most suitable fields of application for the AxC, since the great inherent error resiliency resulting from retrieving models through iterative training algorithms exploiting large datasets [5], [15]. Since one of the main obstacles hindering the widespread adoption of commercial hardware accelerators for these applications is lack of scalability, a significant body of work has been devoted to exploiting the AxC while aiming at reducing their computational demands as well as resource requirements, even though the effort is mainly devoted to NNs rather than DT MCSs.

Concerning the latter, the authors of [16] observed that only a fraction of the data in a given data set really needs the full computational power of a classifier. Thus, they propose to train a set of models with increasing complexity and accuracy rather than a single, complex one, resorting to dynamic reconfiguration during the inference phase to select the appropriate model to apply to a given input. Furthermore, a confidence level for each inference is computed, and if that confidence falls above a certain threshold, the classification process is terminated; otherwise, the inference resorts to more accurate models.

Alternatively, comparators that test feature values against thresholds at each node of a tree can be replaced with approximate ones to reduce the hardware overhead of DT MCSs [2]. Indeed, it is possible to reduce the number of bits required for representing model features by neglecting the least significant ones while keeping the weight of the retained ones unchanged. The removal of parts of the logic needed by comparisons reduces the size of circuits, providing savings both in terms of silicon area and power consumption. Anyway, selecting the number of bits to be neglected for each of the features (and thresholds) to minimize hardware overhead while simultaneously pursuing accuracy loss minimization is quite challenging, as they are conflicting design goals. In [1], the authors addressed this challenge while resorting to Multi-Objective Optimization (MOO). In particular, they exploit the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [17] to carefully select approximate configurations that simultaneously minimize both the accuracy loss and Field Programmable Gate Array (FPGA) Look-Up Tables (LUTs).

Pertaining to NN based systems, state-of-the-art hardware accelerators are tremendously resource intensive due to the massive amount of processing elements needed to effectively accelerate the computations [18]. Multipliers are recognized as the most demanding component in terms of overhead, both in terms of silicon area and power consumption [19]; consequently, a significant number of libraries providing hundreds of implementations of approximate components delivering different trade-offs between error and hardware resource requirements, such as the EvoApproxLib [20], have been proposed. In [3], suitable approximate multipliers are selected from the mentioned library and deployed to convolution layers, while the overall classification error and energy consumption are simultaneously minimized through MOO. Since the impact of approximation on the end quality also depends on the input characteristics, the requirement for accuracy may not be static. Hence, the authors of [4] designed and deployed a dynamically reconfigurable multiplier, allowing the desired accuracy level to be dynamically selected at run time. The deployment is performed while resorting to the ratio between the accuracy loss due to approximation of one specific layer and the original accuracy – namely, the layer significance. The latter is exploited to greedily map convolution layers exhibiting low significance to the highest approximation degree. Furthermore, for the layers that cannot be entirely mapped to the highest approximation degree, the mapping is computed filter-wise.

Regardless of the final application, assessing the error due to the introduced approximation is mandatory, since the approximation usually degrades the final quality of results. Thus, for ACSs in our specific case, a software emulator of

the accelerator, typically executed either on CPUs or GPUs, is performed to measure the introduced error. However, this emulation is typically several orders of magnitude slower than the reference software implementation operating with standard (non-approximate) floating-point or integer arithmetic, since neither CPUs nor GPUs provide support for fast emulation of approximate operations. In this paper, we propose a technique to reduce the computational time needed to measure the accuracy loss. In particular, we select samples from the test data set that are likely to be more sensitive to the approximation. Then, we exploit them for a quicker estimation of the accuracy loss, accelerating the DSE process and allowing for a broader exploration of the design space.

3 Reducing the Effort of the Error Assessment Procedure

During DSE, the classification accuracy is commonly adopted as one of the fitness functions to be evaluated to rank approximate variants. In particular, the classification accuracy loss is either constrained or simultaneously minimized while optimizing performance metrics, respectively, in single or multi objective approaches [1], [3], [4], [11].

Our goal is to reduce the computational time required to assess the impact of perturbations on classification accuracy. This can be achieved either by reducing the size of the test dataset or by using a different error metric. To the best of our knowledge, there is no other error metric to be exploited than accuracy-loss to measure the impact of approximation on ACSs. Hence, we focus on the test dataset; specifically, on test samples whose classification may be more susceptible to approximation.

In the following, we first provide the reader with definitions and notations adopted in the paper, then we focus how we can identify test samples whose classification may be more susceptible to approximation. To this aim, we try to figure out effects of perturbations on CS, giving an insight of AxC as source of perturbation, supporting our hypotheses by reporting preliminary experimental result. Then, we intuitively exploit the susceptibility of test samples to AxC perturbation introducing our ranking method.

3.1 Definitions and notations

CSs are generally differentiated by their data model (for instance, Decision Tree (DT) uses a tree data structure) and they exploit two algorithms, namely training and testing, both based on a pre-labeled dataset $\boldsymbol{D}$. Indeed, the dataset $\boldsymbol{D}$ is partitioned into two different datasets, one used to train the classification model, the other to estimate the accuracy reached by the trained model.

The main objective of a training algorithm is to obtain a model with as high accuracy as possible. So let τ_i be a test sample, e.g., test images for NNs, got from test set $\boldsymbol{T} = \{\tau_0, \tau_1, \tau_2, \cdots, \tau_{M-1}\} \subset \boldsymbol{D}$ and let θ^*_τ be the corresponding class label retrieved from the dataset. Let also $\mathcal{P}(\tau_i)$ be the test algorithm over the sample τ_i, that is the inference on the test-sample τ. Generally speaking, the inference results in the *likelihood vector* $\boldsymbol{\rho} := \mathcal{P}(\tau_i)$, that has as many elements as possible classification outcomes: $\boldsymbol{\rho} = \{\rho_0, \rho_1 \ldots \rho_{N-1}\}$. The j-th element of the latter denotes the *score* that $\mathcal{P}$ assigned to the j-th class for the test-sample τ_i, which can be interpreted as the likelihood (or probability) of τ_i belonging to the j-th class. Hence, it is clear that:

$$\boldsymbol{\rho} : \textstyle\sum_j \rho_j = 1 \text{ with } 0 \leq \rho_j \leq 1 \; \forall j \tag{1}$$

It follows that the predicted class over the sample τ_i, namely θ_{τ_i}, is:

$$\theta_\tau := \arg\max(\boldsymbol{\rho}) \tag{2}$$

Although it might seem superfluous, for clarity it is hereby reported that the likelihood of τ belonging to θ_τ, as given by $\mathcal{P}$, is $\max(\boldsymbol{\rho})$. We denote such likelihood as ρ_{θ_τ} (3).

$$\rho_{\theta_\tau} = \max(\boldsymbol{\rho}) \tag{3}$$

Please bear in mind that, in case a misclassification occurs, $\theta^*_\tau \not\equiv \theta_\tau$; in this case, we denote as ρ_{θ^*} the likelihood that the CS assigned to the *actual* class of τ. Furthermore, we denote as $\boldsymbol{\Theta}$ the set of correctly classified test samples, i.e.,

$$\boldsymbol{\Theta} := \{\tau | \theta^*_\tau \equiv \theta_\tau\} \tag{4}$$

where θ_τ is given by Equation (2) and θ^*_τ is the actual class of τ. Finally, the classification accuracy of $\mathcal{P}$ is given by (5), where $|\cdot|$ denotes the size of a set.

$$\eta = \frac{|\boldsymbol{\Theta}|}{|\boldsymbol{T}|} \tag{5}$$

3.2 Investigating on Classification System Resiliency

We are pretty confident, due to many fundamental results from the scientific literature, that CSs are resilient to error [5]. This property is indeed imprinted by training algorithms, since they exploit data-sets that contain a significantly high numbers of properly distributed pre-labeled samples. Such an error resilience is not only an advantage from a classification perspective, since it gives to input data of CSs some degree of tolerance w.r.t. error, but it is a feature that AxC techniques can exploit to improve the performance of these systems.

Without loss of generality, both in case of input-data error or due to AxC techniques, we can state that a CS is resilient to perturbation. Let $\rho' = \mathcal{P}(\tau'_i) \equiv \mathcal{P}'(\tau_i)$ be the classification outcome due to such a perturbation. Following Equations (2) and (4), we can measure the impact of the perturbation onto resulting prediction accuracy.

$$\eta' = \frac{|\boldsymbol{\Theta}'|}{|\boldsymbol{T}|} \tag{6}$$

Resorting to Equation (2), for each sample τ, we can distinguish three different cases:

Case A: $\arg\max(\boldsymbol{\rho}') \equiv \theta^*_\tau \equiv \arg\max(\boldsymbol{\rho})$, thus the resulting accuracy is not affected w.r.t. $\mathcal{P}$; on the contrary, we may have either

Case B: a misclassification due to the perturbation leads to an accuracy loss, hence: $\arg\max(\boldsymbol{\rho}') \not\equiv \theta^*_\tau \equiv \arg\max(\boldsymbol{\rho})$, or

Case C: the perturbation causes a misclassified sample to be properly classified, i.e., $\arg\max(\boldsymbol{\rho}') \equiv \theta^*_\tau \not\equiv \arg\max(\boldsymbol{\rho})$, that is beneficial to the accuracy of $\mathcal{P}'$.

As a consequence, whether Case A occurs, the accuracy of perturbed CSs is not modified w.r.t. the reference predictor. It follows that CS resiliency must reside in the fact that most samples fall into Case A.

Let us now consider the element-wise difference between likelihood vectors resulting from the reference and the perturbed classifiers, that is:

$$\delta_j = \rho_j - \rho'_j \, , \forall j \tag{7}$$

Clearly, for Case A, this difference must turn out to be negligible and do not have an effect on Equation (2). Conversely, for what pertains to Case B or Case C, Equation (2) gives a different outcome, implying the following:

$$\exists i : \arg\max(\boldsymbol{\rho}) \neq \arg\max(\boldsymbol{\rho} + \boldsymbol{\delta}), \, \boldsymbol{\rho} = \mathcal{P}(\tau_i), \, \boldsymbol{\delta'} = \boldsymbol{\rho} - \mathcal{P}'(\tau_i) \tag{8}$$

Due to the inherent resiliency of CSs, we know that Case A is predominant against Case B and Case C, and hence the value given by Equation (7) does not change the maximum value index of the likelihood vector, so arbitrary $|\delta_j| \leq \epsilon$. Conversely, if the perturbation is strong enough, Cases B and C are possible, so arbitrary $|\delta_j| > \epsilon$. Of course, we do not know much about the distribution of δ_j since it should be analyzed by varying the source of perturbation, i.e., data error distribution and AxC techniques, but we can solely state that, due to manifest resiliency of CSs, it likely does not change the classification outcome when accuracy loss is negligible.

Even though, some considerations are still possible just taking a look at values of $\rho_i \in \boldsymbol{\rho}$. Let us consider $\boldsymbol{\rho} = [\rho_0, \rho_1, \cdots \rho_{N-1}]$, and resorting to Equations (2, 3) we can distinguish three different scenarios, i.e.:

Scenario A: the classification outcome is $\rho_\theta \approx 1, \;\; \rho_j \approx 0 \;\; \forall j \neq \theta$, i.e., the CS is pretty much like sure that τ belongs to the class θ;

Scenario B: the classification outcome is $\rho_\theta \approx \rho_j \approx \frac{1}{2}, \rho_\theta > \rho_j, \;\; j \neq \theta, \, \rho_k \approx 0 \;\; \forall k \neq \theta, \forall k \neq j$, i.e., the CS is uncertain between two classes, to which a very similar probability is associated;

Scenario C: $\rho_\theta \approx \rho_j \approx \frac{1}{N}, \rho_\theta > \rho_j \;\; \forall j \neq \theta$; that is, the probability of being labeled with one class or another is the same; this is clearly the degenerate case w.r.t. Scenario B.

As for Scenario A, the predicted class has a probability value far from other classes and, resorting to Equation (7), the prediction is expected to be less *sensitive* to the considered perturbation. Thus, such a sample τ does not likely contribute to change accuracy. On the contrary, in Scenario B and Scenario C, the predictor $\mathcal{P}$ is quite *sensitive* to the perturbation and $\mathcal{P}'(\tau)$ is likely to result in a different class. Thus, w.r.t. samples τ_i, resiliency of a CSs must derive from the numerous samples τ_i that fall in Scenario A.

In order to measure how resilient is each sample τ w.r.t. predictor $\mathcal{P}$, we can exploit the normalized Gini-Simpson index (9), or Gini impurity, from [21], a well-known metric already widely employed in machine learning (especially during DTs training), or alternatively the min-entropy estimation or the Euclidean distance. The Gini impurity, in particular, is a direct measure of how often a randomly chosen element of a given set would be incorrectly labeled if it was labeled randomly and independently according to the distribution of labels in that set.

$$I_G(\boldsymbol{\rho}) = \tfrac{N}{N-1} \left(1 - \textstyle\sum_{j=0}^{N-1} \rho_i^2\right), \, \boldsymbol{\rho} = \mathcal{P}(\tau), \, \tau \in \boldsymbol{T} \tag{9}$$

It reaches the minimum value of 0 when all elements in the set have the same class label, that is equivalent to our Scenario A, while it reaches the maximum value of 1 when elements in the set are equally distributed among all classes, that is our Scenario C.

Hence, in conclusion, we can state that training algorithms, exploiting the repetitiveness and redundancy of information in the dataset, aim to achieve the optimum on the trained model by maximizing the number of samples falling into Scenario A and, consequently, minimizing – either directly, or indirectly – the loss function for each sample in the given dataset, e.g., the Gini index for the CART algorithm [22].

3.3 AxC as Perturbation Source

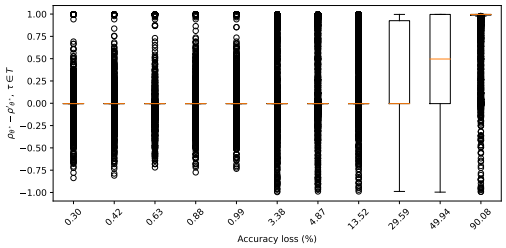
As aforementioned, during DSE of approximate variants of a targeting CS, among others, classification accuracy is considered as a maximizing fitness function. Then, those AxC variants that show an accuracy loss below a predetermined tolerance threshold are taken into account. This inherently implies that, for obtained AxC variants, δ is distributed around zero and its absolute value is strictly correlated to the actual accuracy loss.

Let us now empirically investigate how AxC impact onto CS. To this aim, we set up a preliminary experimental campaign in which we considered both DT MCSs and NNs. As for the former, we considered DT MCSs trained to classify samples taken from the Avila [23] and the Dry Bean [24] datasets, on which 97.69% and 91.84% accuracy is achieved, respectively. As for the latter, we considered LeNet-5 [25] trained to classify images from the Modified National Institute of Standards and Technology (MNIST) benchmark [26], and ResNet-8 [27] targeting images taken from the CIFAR-10 dataset [28]. These NNs exhibit 99.07% and 86% accuracy, respectively. Besides, we resort to state-of-the-art-methodologies to design ACSs for each of the mentioned systems, while pursuing accuracy-loss and hardware requirements minimization. Kindly note that these systems are part of the test bed for subsequent in-depth analysis; therefore, details concerning models, dataset, and ACS design methodologies being used are omitted here, and they will be provided to the reader in Section 4.

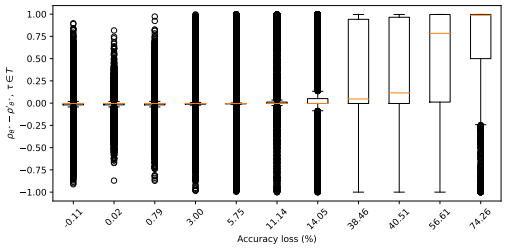
In this preliminary experimental campaign, given a test set T, a predictor $\mathcal{P}$ and a set of ACSs, we computed, for each of the samples, the mean absolute perturbation induced by approximation on each score of the classes as in (10), that is the mean of the absolute element-wise difference between the scores that $\mathcal{P}$ and an ACS $\mathcal{P}'$ assigns to a sample τ, for each sample $\tau \in T$, with N being the number of possible classification outcomes, i.e., the number of classes. Please note that $E(\tau)$ (10) is bounded due to (1).

$$E(\tau) = \tfrac{1}{N} \textstyle\sum_{i=0}^{N-1} |\rho_i - \rho'_i| = \tfrac{1}{N} \sum_{i=0}^{N-1} |\delta_i| \tag{10}$$

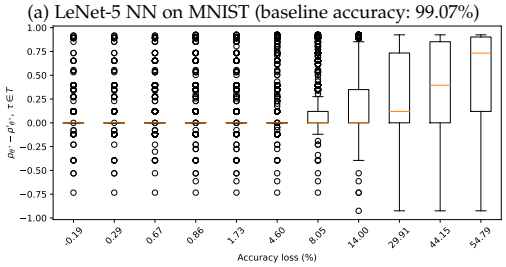
Figure 1 and Figure 2 show the element-wise error between the scores that $\mathcal{P}$ and an ACS $\mathcal{P}'$ assigns to a sample $\tau \in T$ (namely δ as defined in Equation (7)), and the mean of the absolute element-wise difference between the scores $E(\tau)$, as defined by (10), respectively. As the reader can observe, although the error is almost always close to zero for all ACSs, its variation range becomes larger as the degree of approximation increases, hence the loss of accuracy increases. Moreover, the higher the baseline



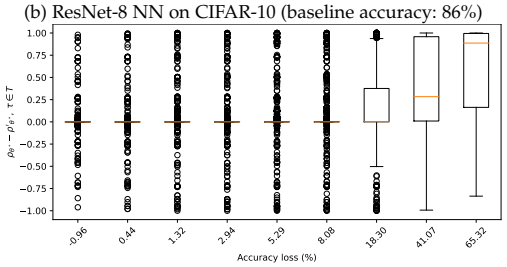
(a) LeNet-5 NN on MNIST (baseline accuracy: 99.07%)



(b) ResNet-8 NN on CIFAR-10 (baseline accuracy: 86%)

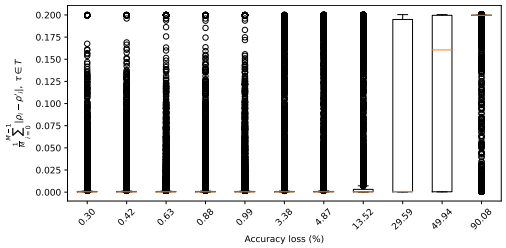


(c) 5-DT MCS NN on Avila (baseline accuracy: 97.69%)

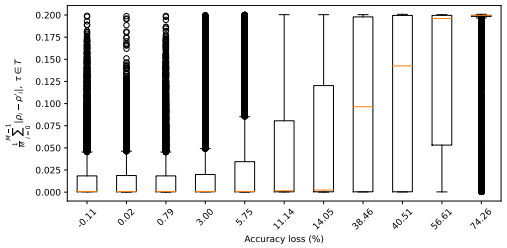


(d) 13-DT MCS NN on Dry Bean (baseline accuracy: 91.84%)

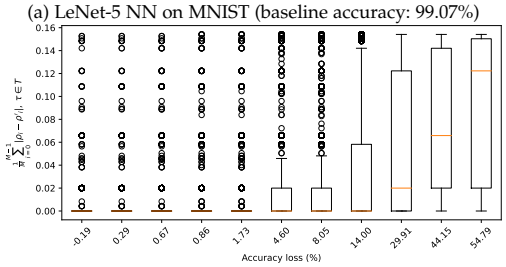
Fig. 1: Box-and-whisker plots for $\rho_{\theta*} - \rho'_{\theta*}$, for different ACSs, while considering $\tau \in \boldsymbol{T}$



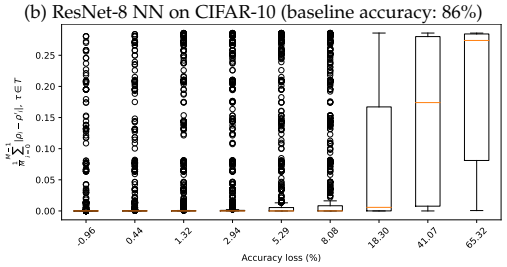
(a) LeNet-5 NN on MNIST (baseline accuracy: 99.07%)



(b) ResNet-8 NN on CIFAR-10 (baseline accuracy: 86%)



(c) 5-DT MCS NN on Avila (baseline accuracy: 97.69%)



(d) 13-DT MCS NN on Dry Bean (baseline accuracy: 91.84%)

Fig. 2: Box-and-whisker plots for the mean of the absolute element-wise difference between the scores $E(\tau)$, as defined by (10), for different ACSs, while considering $\tau \in \boldsymbol{T}$

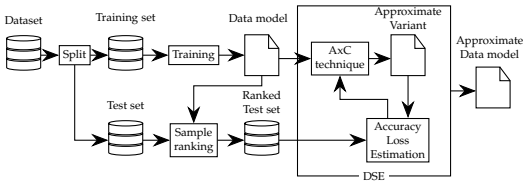


Fig. 3: Dataflow of a DSE approach involving the test set ranking proposed in this work.

accuracy of the classifier, the lower the value of E, as can be seen by comparing Figure 2a, Figure 2b, Figure 2c, and Figure 2d. In fact, classifiers with higher baseline accuracy tend to have a narrower range of variation for Equation (10).

3.4 Test dataset ranking

Taking considerations of the above, when we need to estimate the accuracy of $\mathcal{P}'$ we can exploit, instead of (6), the accuracy formerly computed over the reference predictor $\mathcal{P}$. Indeed, we proved that Case A does not contribute to modifying accuracy of $\mathcal{P}'$, while Case B and Case C do cause accuracy to change its value. Hence, we need to identify which samples τ are most likely to fall in these two cases.

Since the approximation evenly perturbs the vector $\boldsymbol{\rho}$ in its entirety, rather than a single value ρ_i, and since Equation (1) must hold, given a sample τ_i, if some ρ_i decreases/increases due to the approximation, some others ρ_j, $j \neq i$ increase/decrease accordingly. Consequently, whether the difference between ρ_θ (3) and other scores ρ_j, $j \neq \theta$ is negligible, the error induced by approximation is likely to impact the classification outcome, possibly leading to misclassification (Scenario B and Scenario C). On the other hand, whether the difference in terms of score between ρ_θ and the others is significantly high, the approximation is likely to cause no impact (Scenario A).

Based on the metric as defined in Equation (9), we

rank test samples belonging to $\boldsymbol{\Theta}$ – i.e., test samples being correctly classified by $\mathcal{P}$, as defined in (4) – and test samples belonging to $\boldsymbol{T} \setminus \boldsymbol{\Theta}$ – i.e., test samples being misclassified by $\mathcal{P}$. In Figure 3 we sketched the dataflow of a DSE campaign over a machine learning datamodel that exploits the test dataset ranking.

The impurity-based ranking procedure is described in Algorithm 1. The procedure takes as input the predictor $\mathcal{P}$ and the test dataset $\boldsymbol{T}$, and produces partitions of $\boldsymbol{T}$ for correctly classified and misclassified test samples, which are empty at the beginning (lines 1 and 2) and filled as inference is performed over $\tau \in \boldsymbol{T}$. Specifically, for each $\tau \in \boldsymbol{T}$, the actual class θ_τ is extracted from τ itself, the likelihood vector $\boldsymbol{\rho}$ is computed using $\mathcal{P}(\tau)$. Then, the impurity index $I_G(\boldsymbol{\rho})$ as defined by Equation (9) is computed, as well as the predicted θ_τ. Finally, τ is assigned either to $\boldsymbol{C}$ or $\boldsymbol{M}$, based on the predictor result. During the DSE, in order to quickly assess the impact of the approximation and possibly quickly identify and discard approximate solutions with high loss values, it is convenient to perform the inference phase starting from test samples exhibiting higher impurity, i.e., those for which the classification is supposed to be more sensitive to perturbations, and hence to approximation. Therefore, samples belonging to $\boldsymbol{C}$ are sorted in descending order based on their impurity index. Conversely, as far as misclassified samples are concerned, the approximation may be beneficial for those samples that have a lower impurity, eventually leading to a correct classification. Therefore, samples belonging to $\boldsymbol{M}$ are sorted in ascending order based on their impurity index.

Please note that the dataset ranking procedure has to be performed only once, and, through Equation (5) it also provides the actual baseline accuracy of the non-approximate classifier on the test dataset.

Algorithm 1 Impurity-based dataset ranking procedure

```
1:  procedure SAMPLERANKING(P, T)
2:      C ← ∅
3:      M ← ∅
4:      for τ ∈ T do
5:          θ* ← classOf(τ)
6:          ρ ← P(τ)
7:          θ ← arg max(ρ)
8:          i ← I_G(ρ)
9:          if θ*_τ ≡ θ_τ then
10:             C ← C ∪ {(τ, i)}
11:         else
12:             M ← M ∪ {(τ, i)}
13:         end if
14:     end for
15:     C ← sortDescending(C)
16:     M ← sortAscending(M)
17:     return C, M
18: end procedure
```

3.5 Optimizing accuracy-loss estimation

Our rank-based loss estimation approach is formalized in the Algorithm 2. Instead of computing accuracy as in Equation 6, we want to take advantage of the fact that the samples are sorted by susceptibility to approximation, and then estimate the accuracy value, $\hat{\eta}'$ on a subset of the testset. To this aim, the estimation procedure needs the ACS $\mathcal{P}'$, the $\boldsymbol{C}$ and $\boldsymbol{M}$ subsets of the test dataset, as computed by Algorithm 1, the baseline accuracy η of the reference predictor $\mathcal{P}$, and the maximum allowed accuracy loss $\nu_{\%}$. Please note that the baseline accuracy η parameter of Algorithm 1 is superfluous, since $\eta = \frac{|\boldsymbol{C}|}{|\boldsymbol{C}|+|\boldsymbol{M}|}$. It is hereafter used solely for readability purpose.

Algorithm 2 first computes the maximum number of allowed mispredictions to meet the constraint on the accuracy loss, then it performs the inference starting from samples belonging to $\boldsymbol{M}$, which are those misclassified by $\mathcal{P}$. In case a test sample is correctly classified by the ACS $\mathcal{P}'$, the number of misclassifications is decreased, and, as a result, the maximum number of allowed misclassifications is increased accordingly, as more mispredictions can be tolerated to meet accuracy constraint while considering samples $\tau \in \boldsymbol{C}$. Furthermore, the algorithm counts the number of misclassified samples, and breaks the inference on $\boldsymbol{M}$ as soon as that count exceeds a certain threshold, which is given by the β parameter. In fact, samples in $\boldsymbol{M}$ are evaluated starting from those with lower impurity – i.e., those for which approximation is likely to be beneficial leading to a correct classification – and as the procedure examines samples respecting the order given by impurity, this likelihood decreases. Therefore, it is reasonable to assume that once b consecutive samples have been misclassified, so will the remaining ones.

The Algorithm behaves almost the same regarding samples belonging to $\boldsymbol{C}$ – i.e., those correctly classified by $\mathcal{P}$ – with minor differences. In case the ACS outcome does not match the expected one, the number of mispredictions is increased, and whether it exceeds the threshold, the ACS $\mathcal{P}'$ cannot meet the accuracy constraints; thus, the procedure breaks. As done for samples in $\boldsymbol{M}$, the algorithm counts the number of consecutive correctly classified samples, and it stops sooner whether this count exceeds a given threshold, which corresponds to the α parameter of the estimation procedure. These samples, in fact, are sorted in descending order based on the impurity degree, and since it proceeds with samples with lower impurity – that are less susceptible to approximation – it is easy to expect that, at some point, the ACS will stop misclassifying samples belonging to the $\boldsymbol{c}$ set. However, stopping the procedure prematurely at the first correctly classified sample is dangerous regarding the confidence of the obtained estimate. Furthermore, as far as samples belonging to $\boldsymbol{C}$, resetting the counter of consecutive correct classifications to zero in case a misclassification occurs is pretty constrictive, and since it greatly affects the number of samples on which the inference is computed, it also has a significant impact on computation time. We deem that the count could be reduced, for example by halving it, instead of resetting the count value to zero, trading-off the confidence on the loss value for quicker estimations. The reduction of this count can be controlled using the γ parameter of the algorithm.

Note that, $\hat{\eta}' \leq \eta'$, being $\hat{\eta}'$ computed over a subset of $\mathbf{T}$ as shown in the Algorithm 2. So, in case of accuracy loss is less than ν, we are able to provide an underestimation, which confidence is guaranteed to be as close to 100% as α is high. Conversely, in case the accuracy loss is less than ν, the confidence is always 100%. Moreover, by setting α and β very high, more samples are considered during the loss estimation procedure, and, at the limit, it is possible to make

sure that the entire dataset is considered. The same effect is obtained by setting γ very high. Furthermore, setting $\gamma \equiv |\boldsymbol{T}|$ equals setting $\alpha \leftarrow 0$ at line 24, resetting the counter of consecutive correct classifications, resetting the counter for consecutive correct classifications.

Algorithm 2 Accuracy-loss estimation procedure

1: **procedure** ESTIMATELOSS($\eta_{\%}, \nu_{\%}, \alpha, \beta, \gamma, \mathcal{P}', \boldsymbol{C}, \boldsymbol{M}$)
2: $maxMiss \leftarrow \lfloor (100 - \eta_{\%} + \nu_{\%}) \cdot \frac{|\boldsymbol{C}|+|\boldsymbol{M}|}{100} \rfloor$
3: $miss \leftarrow 0$
4: $b \leftarrow 0$
5: **for** $\tau \in \boldsymbol{M}$ **do**
6: $\theta^* \leftarrow$ class_of(τ)
7: $\boldsymbol{\rho} \leftarrow \mathcal{P}'(\tau)$
8: **if** $\theta^* \equiv \arg\max(\boldsymbol{\rho})$ **then**
9: $miss \leftarrow miss - 1$
10: $b \leftarrow 0$
11: **else**
12: $b \leftarrow b + 1$
13: **end if**
14: **if** $b \geq \beta$ **then**
15: **break**
16: **end if**
17: **end for**
18: $a \leftarrow 0$
19: **for** $\tau \in \boldsymbol{C}$ **do**
20: $\theta^* \leftarrow$ class_of(τ)
21: $\boldsymbol{\rho}_\tau \leftarrow \mathcal{P}'(\tau)$
22: **if** $\theta^* \not\equiv \arg\max(\boldsymbol{\rho})$ **then**
23: $miss \leftarrow miss + 1$
24: $a \leftarrow \lfloor a/\gamma \rfloor$
25: **else**
26: $a \leftarrow a + 1$
27: **end if**
28: **if** $miss \geq maxMiss$ **or** $a \geq \alpha$ **then**
29: **break**
30: **end if**
31: **end for**
32: **return** $\frac{100 \cdot miss}{|\boldsymbol{C}|+|\boldsymbol{M}|}$
33: **end procedure**

3.6 Final remarks

In the above, we appreciated how the training algorithms of CSs lead to trained models that are robust against perturbations. In particular, based on a probabilistic inference, we can assume that the majority of samples belonging to the test set are assigned likelihood vectors that fall onto Scenario A. A side result, then, concerns the direct measure of the resilience of a CS that we can obtain from the data set, that is, how the impurity index is distributed over the entire data set w.r.t. the trained model. Intuitively, models characterized by impurity index values distributed tightly close to zero are more resilient than ones characterized by impurity index values distributed more uniformly. For these samples, perturbations due to approximation would rarely impact the classification outcome. Resorting to models and datasets introduced in Section 3.3, we empirically investigate on the distribution of the impurity index, given a test dataset $\boldsymbol{T}$ and a classifier $\mathcal{P}$. Specifically, we computed the Gini-impurity (9) for each of the sample $\tau \in \boldsymbol{T}$, distinguishing between correctly classified and misclassified test samples. Figure 4 reports results: the left-hand plots in figures 4a, 4b, 4c and 4d refer to correctly classified test samples – $\tau \in \boldsymbol{\Theta}$ –, while the right-hand ones refer to misclassified samples.

As the reader can observe, for correctly classified test samples (left-hand plots), the Gini-impurity (9) concentrates very close to zero, confirming our assumption concerning the majority of samples of the test set being assigned likelihood vectors that fall onto Scenario A. Points beyond the whiskers on the box-plots, conversely, pinpoint samples for which the likelihood vectors that fall onto either Scenario B or Scenario C. In the latter two cases, however, the approximation could have either negative effects – a sample previously classified correctly could be misclassified by the ACS – or no effect at all – a sample continues to be misclassified.

As for misclassified test samples (right-hand plots), for most of the considered datasets and classifiers the Gini-impurity values (9) distribute undeniably far from zero, around the center-value $I_G(\boldsymbol{\rho}) \approx \frac{1}{2}$. This happens whether $\rho_\theta \approx \rho_j \approx \frac{1}{2}, \rho_\theta < \rho_j,\ j \neq \theta,\ \rho_k \approx 0\ \forall k \neq \theta, \forall k \neq j,$ i.e., the CS is uncertain between two classes, to which a very similar probability is given, however it determined the wrong class as the winner. In this case, the approximation can even be beneficial, since it could favor the correct class to the detriment of the current wrong outcomes. The rightmost plot in Figure 4d, anyway, depicts a Gini-impurity distribution close to zero for misclassified test samples as well. This is dual to Scenario A: $j \neq \theta : \rho_j \approx 1, \rho_\theta \approx 0$, which means the CS is pretty much like sure that τ belongs to the class j, albeit j is the wrong class. As for Scenario A, perturbations due to approximation would rarely impact the classification outcome; thus, it is unlikely that approximation would lead to improved accuracy.

4 EVALUATION

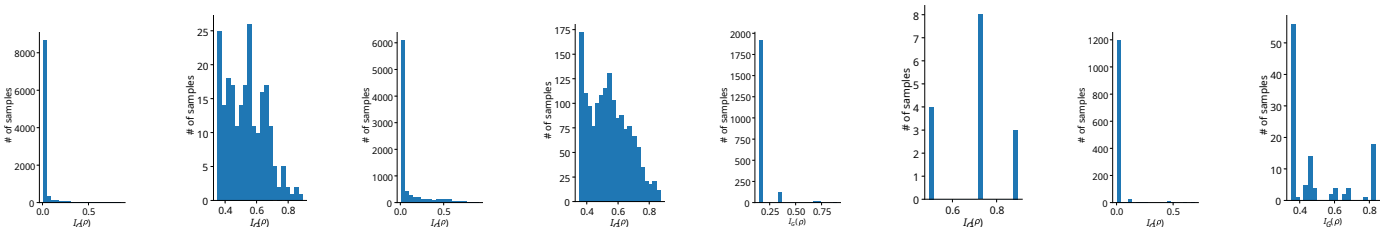
In this Section, we evaluate our approach while designing ACS involving two quite common machine-learning models, namely DT MCSs and NNs. We aim to empirically prove that, when compared to methods involving a full evaluation of the dataset for accuracy loss estimation, our method can deliver equally good solutions in less time. This is achieved through a faithful estimation of the accuracy loss performed while considering only test samples that are actually affected by approximation. Furthermore, we empirically prove that, when a limited computational-time budget is considered, the rank-based approach can provide even better solutions when compared to approaches considering the full dataset.

Before providing evidence supporting the above statements in Section 4.2, we first provide full details concerning the experimental setup in Section 4.1. In particular, we will discuss how reference (non-approximate) CSs, as well as ACS, have been designed while resorting to state-of-the-art methodologies, including a brief discussion concerning parameters influencing the DSE and Algorithm 2, i.e., the accuracy-loss estimation procedure.

4.1 Experimental setup

In our analysis, we considered five different datasets, addressing the classification task while resorting to seven different CSs. As mentioned, we considered both DT MCSs and NNs, which design is detailed in Section 4.1.1 and Section 4.1.2, respectively.

Before going deeply into discussing the design of ACSs, we deem it worth noticing that, despite the evident differences in these two CSs, their design has some similarities. For instance, concerning the DSE it is worth noticing that



(a) LeNet-5 on MNIST (baseline accuracy: 99.07%)

(b) ResNet-8 on CIFAR-10 (baseline accuracy: 86%)

(c) 5-DT MCS on Avila (baseline accuracy: 97.69%)

(d) 13-DT MCS on Dry Bean (baseline accuracy: 91.84%)

Fig. 4: Distribution of the Gini impurity across test samples for different predictors. The left-hand plots refer to correctly classified test samples, while the right-hand ones refer to misclassified test samples.

Dataset	Instances	Attributes	Classes	Predictor	Baseline accuracy
Avila	20867	10	12	5-DT MCS	97.69%
Dry Bean	13611	17	7	13-DT MCS	91.84%
Spambase	4601	53	2	20-DT MCS	92.82%
MNIST	60000	N/A	10	LeNet-5	99.07%
CIFAR-10	60000	N/A	10	ResNet-8	84.31%
				ResNet-14	86%
				ResNet-24	85.78%

TABLE 1: Dataset and models being considered for evaluation

there are some parameters that depend neither on the particular classification model nor on the dataset, although they definitely affect both the result quality and computational time. In particular, for the NSGA-II, that is exploited to cope with DSE when targeting both DT MCSs and NN, those of major influence are the initial population size, the number of generations, and the mutation and crossover probabilities. According to the guidelines and suggestions from [29], we have succeeded in achieving a good configuration for the NSGA-II through successive attempts. Hence, during the experimental phase, several campaigns were conducted, during which the configuration parameters of the NSGA-II were modified several times, aiming at Pareto frontiers that were sufficiently diversified and populous.

As far as parameters governing the behavior of Algorithm 2 – i.e., α, β and γ – they undeniably impact the number of samples involved in the accuracy loss estimation. We evaluated through successive attempts that $\alpha = 0.07 \cdot |\boldsymbol{T}|$, $\beta = 0.5 \cdot |\boldsymbol{T}|$, and $\gamma = 2$ can provide a good tradeoff between error estimation confidence and speed-up, as shown later on in this Section. A further fine-tuning of such parameters would lead to negligible enhancmenents, so it is not worth a deeper discussion in this paper.

As far as the effectiveness of our approach – i.e., proving that , when compared to methods involving a full evaluation of the dataset for accuracy loss estimation, our method can deliver equally good solutions in less time – we employed the same configuration for the DSE both when the accuracy assessment is done on the whole dataset and when it is done using our approach, for a rigorous and fair comparison of results. Such configurations will be discussed in the following sections. Conversely, in oder to prove that our approach allows for superior solutions whether a limited computational-time budget is considered, we exploited the running performance metric and early-termination criterion from [30]. Last, in order to discard solutions with a significantly high accuracy loss, we defined a maximum accuracy loss threshold, which is set to 5%.

4.1.1 Decision-tree based classifiers

For what pertains to DT MCSs, classifiers can be designed starting from the reference dataset through different machine-learning frameworks, such as KNIME [1] or Scikit-learn [31]. Both describe classifiers using the Predictive Model Markup Language (PMML), which is an XML-based predictive model interchange format that makes use of the IEEE 754 double-precision floating-point representation for feature values and thresholds at each node of the trees. In the following, without loss of generality, we resort to the Scikit-learn [31] for training DT MCSs and, as reported in Table 1, we considered three different datasets: the Avila [23], the Dry Bean [32] and the Spambase [33] datasets, that are all open-source and freely available through the UC Irvine Machine Learning Repository[2]. The Avila data set has been extracted from 800 images of the "Avila Bible", a XII century giant Latin copy of the Bible, and the prediction task consists of associating each pattern to one of 12 copyists. The dataset consists of 20867 test samples, each of which is composed of 10 attributes encoded as floating-point. We addressed the classification task by training a DT MCS consisting of 5 decision trees that reached 97.69%. The Dry Bean dataset is a collection of 13611 images of 7 different registered dry beans taken with a high-resolution camera, from which a total of 16 features were obtained, both integer and floating-point, through segmentation and feature extraction stages. The task is to distinguish between the seven different registered varieties of dry beans with similar features. To cope with this task, we trained a CS consisting of 13 decision trees, providing 91.84% accuracy. The third dataset we consider for DT MCS is the Spambase dataset, which consists of 4601 samples and involves recognizing spam and non-spam emails. Distinguishing between these two classes is to be done by evaluating 53 different features, both integer and floating-point. To deal with this task with reasonably high accuracy, we trained a DT MCS consisting of 20 decision trees exhibiting 92.82% accuracy.

Pertaining to the approximation, we resort to the approach discussed in [1], [2]. It considers the DT MCS as a black box, taking the PMML encoding of the predictor and exploiting the bit-width reduction AxC technique targeting model features' representation to introduce approximation. In essence, the approach from [1] neglects the least significant bits of features while keeping the weight of the retained bits unaltered. This leads to a reduction

1. https://www.knime.com/
2. https://archive.ics.uci.edu/

Dataset	Predictor	Vars.	Solution Space	Pop.Size	Generations	Max.Time
Avila	5-DT MCS	10	$\approx 2 \cdot 10^{17}$	20	31	3h
Dry Bean	13-DT MCS	17	$\approx 5 \cdot 10^{27}$	35	59	5h
Spambase	20-DT MCS	53	$\approx 5 \cdot 10^{98}$	100	151	5h

TABLE 2: MOO instances and corresponding NSGA-II configuration parameters for DT MCS-based ACSs

in the size of circuits due to the removal of parts of the logic needed by comparisons. Finally, MOO, that addressed using the NSGA-II [17], is exploited to search for ACS providing the best trade-offs between hardware overhead and classification accuracy. Regarding this very last point, the configuration parameters for the NSGA-II have been set according to the guidelines and suggestions from [29]. Besides the population size and the maximum number of generations, we report the maximum allowed runtime of the NSGA-II to be used as early-termination criterion in Table 2. Furthermore, albeit not reported in the mentioned Table, we set the crossover and mutation probability to 0.8 and 0.1, respectively.

4.1.2 Neural Networks

As far as NNs are concerned, reference networks can be designed while resorting to state-of-the-art training frameworks, such as TensorFlow [34] or pyTorch [35]. The latter frameworks leverage 64-bit double-precision floating-point to represent synaptic weights, biases, and activations while performing both the training and inference phases. The use of compression techniques, such as quantization, to reduce NN model size is pretty common.

We resort to the TensorFlow [34] framework, leveraging 8-bit Quantization-Aware Training (QAT) features provided by TensorFlow-LITE. The QAT entails quantization during the training phase, clipping and rescaling weights and activations while leveraging quantization thresholds. Hence, the network is trained with simulated quantization. As reported in Table 1, our evaluation involves four different NNs, that are LeNet-5 [25], ResNet-8, ResNet-14, and ResNet-24 [27]. The LeNet-5 Convolutional Neural Network (CNN) has been trained to classify images from the MNIST dataset of hand-written digits [26], on which it exhibits 99.07% accuracy, while the ResNet-8, ResNet-14, and ResNet-24 CNNs, instead, have been trained while targeting images taken from the CIFAR-10 dataset [28], which consists of 60 thousand RGB images, each belonging to one among ten classes. The networks exhibit 84.31%, 86% and 85.78% accuracy on the mentioned dataset, respectively.

Concerning approximation, we resort to the approach from [3], which replaces accurate convolution layers within NNs using approximate ones. Specifically, for each convolution layer, a suitable approximate multiplier is selected from the EvoApprox8b library [20] while the overall classification error – i.e., accuracy loss – and energy consumption are simultaneously minimized through a MOO. The MOO problem is addressed while resorting to the NSGA-II heuristic [17]. We deploy approximate multipliers taken from the 2022 version of the EvoApproxLib-Lite library of approximate circuits, whose error characterization and hardware overhead are reported in Table 3, for the reader convenience.

Furthermore, as done in [3], we also resort to the tuning procedure involving learned weights, in order to recover the accuracy loss. The latter procedure exploits a *weight mapping function*, which is precalculated offline for each approximate multiplier. Given an approximate multiplier M, for each weight w, a weight w' is determined as

$$\arg\min_{w' \in W} \sum_{a \in I} |M(a, w') - a \cdot w| \; \forall w \in W \qquad (11)$$

i.e., the new weight w' is determined such that it minimizes the sum of absolute differences between the output of the approximate and accurate multiplication over all inputs. Finally, besides the accuracy loss, the DSE is driven by the power consumed by the ACS, that is to be minimized. As done in [3], we estimate it as the sum of power-consumption of each multiplier being deployed in a given layer, multiplied the number of operations needed to compute that layer.

Pertaining to the configuration parameters for the NSGA-II, as mentioned, they have been set according to the guidelines and suggestions from [29]. The population size and the maximum number of iterations, as well as the maximum allowed runtime, are reported in Table 4. Furthermore, we once again 0.8 and 0.1 proved to be good setups values respectively for the crossover and mutation probability.

Circuit	MAE (%)	AWCE (%)	MRE (%)	Power (mW)	Area (μm^2)
1KV6	0.00	0.00	0.00	0.425	729.8
1KV8	0.0018	0.0076	0.28	0.422	711.0
1KV9	0.0064	0.026	0.90	0.410	685.2
1KVA	0.019	0.075	2.53	0.391	641.1
1KVM	0.049	0.20	2.40	0.369	652.8
1KVP	0.051	0.21	2.73	0.363	635.0
1KVQ	0.056	0.25	3.64	0.351	599.8
1KX5	0.15	0.69	8.93	0.289	543.0
1KXF	0.34	1.37	15.72	0.237	482.4
1L2J	0.081	0.39	4.41	0.301	558.9
1L2L	0.23	1.16	12.26	0.200	411.6
1L2N	0.52	2.66	27.44	0.126	284.9
1L12	3.08	12.30	135.77	0.052	172.2

TABLE 3: Error characterization and hardware requirements for approximate circuits taken from the EvoApproxLib-Lite library, as reported in [20].

Dataset	Predictor	Vars.	Solution Space	Pop.Size	Generations	Max.Time
MNIST	LeNet-5	5	$\approx 2.48 \cdot 10^5$	10	17	1h
CIFAR-10	ResNet-8	11	$\approx 7.43 \cdot 10^{11}$	25	31	12h
	ResNet-14	14	$\approx 1.28 \cdot 10^{15}$	30	37	12h
	ResNet-24	21	$\approx 4.60 \cdot 10^{22}$	40	57	12h

TABLE 4: MOO instances and corresponding NSGA-II configuration parameters for NN-based ACSs

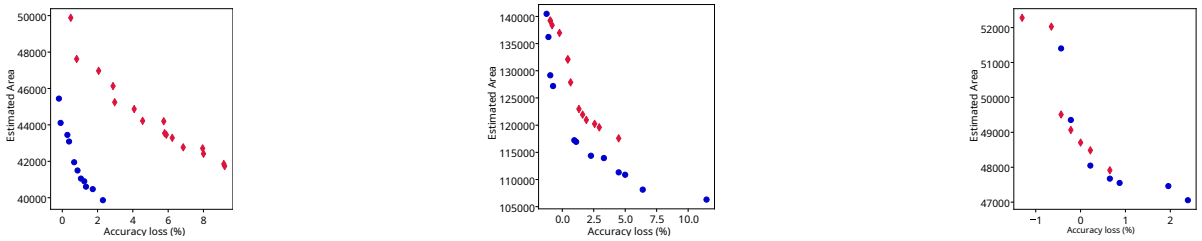
4.2 Experimental results

As mentioned, we considered both DT MCSs and NNs as well as different datasets, for our analysis. In addition, besides collecting DSE results, we also collected the estimation error and the number of test samples exploited during the estimation made through Algorithm 2. As for the estimation error, in particular, it is computed as the difference between the accuracy loss measured on the entire test dataset and its estimation as provided by Algorithm 2.

Regarding the gain in time, as previously stated, we documented the quantity of samples utilized for each estimate

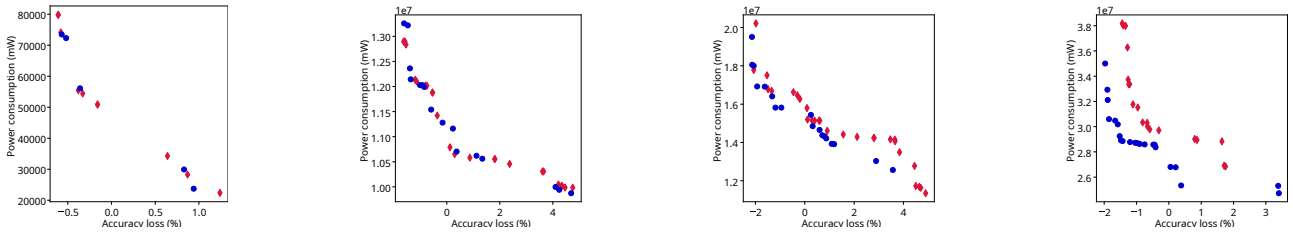
Dataset	Test Samples	Predictor	Avg.Instances	Comp.Full	Comp.Rank	Avg.Speed-up
Avila	2086	5-DT MCS	119	1454	85	≈17
Dry Bean	1361	13-DT MCS	179	2181	311	≈7
Spambase	460	20-DT MCS	46	2988	288	≈10
MNIST	10000	LeNet-5	790	96	7	≈12
CIFAR-10	10000	ResNet-8	2068	2951	610	≈4
		ResNet-14	1929	2891	558	≈5
		ResNet-24	1862	2893	538	≈5

TABLE 5: Speed-up resulting from Algorithm 2



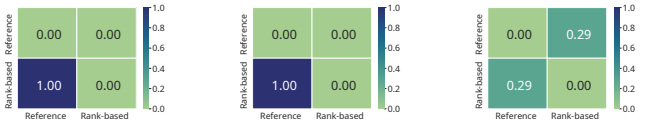
(a) 5-DT MCS on Avila (b) 13-DT MCS on Dry Bean (c) 20-DT MCS on Spambase

Fig. 5: Comparison of results for DT MCS-based ACSs. ♦ denote the Pareto front obtained through exhaustive dataset evaluation, while ● results from the rank-based evaluation approach as reported in Algorithm 2



(a) LeNet-5 on MNIST (b) ResNet-8 on CIFAR-10 (c) ResNet-14 on CIFAR-10 (d) ResNet-24 on CIFAR-10

Fig. 6: Comparison of results for NN-based ACSs. ♦ denote the Pareto front obtained through exhaustive dataset evaluation, while ● results from the rank-based evaluation approach as reported in Algorithm 2



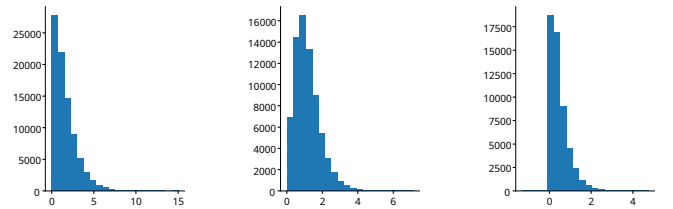
(a) 5-DT MCS on Avila (b) 13-DT MCS on Dry Bean (c) 20-DT MCS on Spambase

Fig. 7: Coverage index (12) for DT MCS-based ACSs.

(a) LeNet-5 on MNIST (b) ResNet-8 on CIFAR-10 (c) ResNet-14 on CIFAR-10 (d) ResNet-24 on CIFAR-10

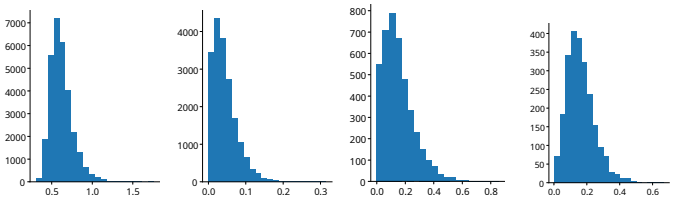
Fig. 8: Coverage index (12) for NN-based ACSs.

in the DSE. These are illustrated in plots of Figure 11 and Figure 12. In most cases, the number of samples considered during estimation is quite small compared to the size of the test dataset. This allows faster estimation, resulting in an average speed increase of up to approximately 18 times, as shown in Table 5. In the mentioned Table, we also reported



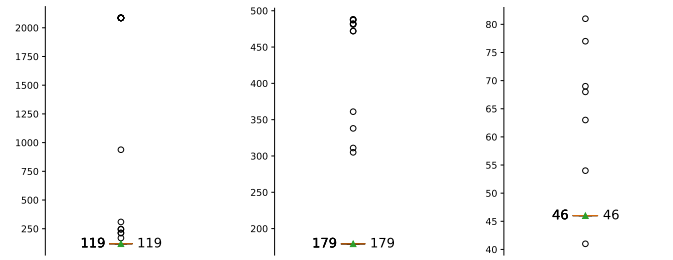
(a) 5-DT MCS on Avila (b) 13-DT MCS on Dry Bean (c) 20-DT MCS on Spambase

Fig. 9: Estimation-error for DT MCS-based ACSs.



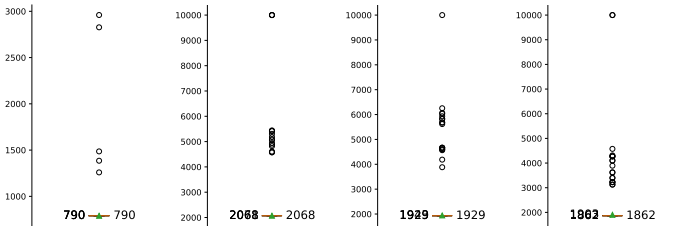
(a) LeNet-5 on MNIST (b) ResNet-8 on CIFAR-10 (c) ResNet-14 on CIFAR-10 (d) ResNet-24 on CIFAR-10

Fig. 10: Estimation-error for NN-based ACSs.



(a) 5-DT MCS on Avila (b) 13-DT MCS on Dry Bean (c) 20-DT MCS on Spambase

Fig. 11: Number of evaluated test samples during the DSE for DT MCS-based ACSs.



(a) LeNet-5 on MNIST (b) ResNet-8 on CIFAR-10 (c) ResNet-14 on CIFAR-10 (d) ResNet-24 on CIFAR-10

Fig. 12: Number of evaluated test samples during the DSE for NN-based ACSs.

computational time required by both the approaches, which have been measured on a computing system equipped with 16-cores/32-threads AMD Ryzen 9550, an NVIDIA A5000 GPU, and 64GB of RAM.

Figure 5 and Figure 6 report Pareto fronts resulting from DSE while designing DT MCSs and NNs. In these, time budgest reported in Table 2 and Table 4 limits the computational time for the DSE. In the these figures, ♦ denote ACSs provided by exhaustive test dataset evaluation, while • refer to those resulting from our rank-based evaluation approach, as in Algorithm 2. Please kindly note that while during the DSE the Algorithm 2 is exploited to shrink the dataset for error estimation, at the end of the DSE the loss is recomputed on the whole dataset and plotted in the mentioned figures. At first glance, for most of the classifiers no significant differences can be easily spotted by observing Figure 5 and Figure 6. This happend whether both the rank-based and the evaluation performed on the whole dataset are able to reach a good estimation of the Pareto front before the allowed time budget. Anyway, it is worth noticing that the Pareto front resulting from Algorithm 2 outperforms that resulting from full dataset evaluation in Figure 5a, Figure 5b,Figure 6c and Figure 6d. This is simply explained by the larger portion of the design space that a reduced fitness computation-time allows, possibly leading to better solutions. It is worth noticing that for the 13-DT DT MCS dataset classifying samples from the Dry Bean (Figure 5b), as well as ResNet-24 classifying images from the CIFAR-10 dataset (Figure 6d, neither of the DSEs reached completion within the given time budget. Nevertheless, it is easy to recognize that the results provided by the rank-based approach are superior, due to the larger portion of the design space being explored.

Whenever no evident difference in Pareto fronts can be observed in the figures, to relieve us of any doubt, we also resort to the *Coverage of two sets* metric [36] that provides a comparative method to determine which between two candidate fronts – resulting from a heuristic such as the NSGA-II – is closer to the actual Pareto front. Let A and B be two sets of solutions for a MOO problem, the $\mathcal{C}(A,\ B)$ maps the pair $(A,\ B)$ to the interval $[0, 1]$ as specified in (12).

$$\mathcal{C}(A, B) := \frac{|\{\forall \beta \in B;\ \exists\ \alpha \in A : \alpha \succeq \beta\}|}{|B|} \tag{12}$$

Whether $\mathcal{C}(A,\ B)$ equals to 1, it means that all points in B are dominated by, or they are equal to, points in A. On the contrary, when no point in B is covered – i.e., dominated – by any points in A, the $\mathcal{C}(A,\ B)$ equals to 0. Kindly note that the expression $\alpha \succeq \beta$ means "α *covers* β", i.e., that the solution α *dominates* the solution β or they are *the same*.

Since $\mathcal{C}(A, B) \neq \mathcal{C}(B,\ A)$, both have to be computed when comparing Pareto fronts [36]. Hence, we computed $\mathcal{C}$ between the Pareto-fronts resulting from exhaustive test dataset evaluation and those resulting from our rank-based approach (and vice versa), reporting results in Figure 7 and Figure 8 respectively for DT MCS and NN based ACSs. The coverage index (12) further confirms our hypothesis: low values of $\mathcal{C}(A, B)$ and $\mathcal{C}(B,\ A)$ mean there is no manifest dominance relationship between the two sets of solutions. This empirically proves that, when compared to methods involving a full evaluation of the dataset for accuracy loss estimation, our method can deliver equally good solutions.

This behavior is rather easily explained. In fact, as evidenced by the plots in Figure 9 and Figure 10, the estimation error, for solutions that exhibit an accuracy loss lower than the given threshold, is really low for both decision tree-based and neural network-based classifiers, regardless of the particular classification problem.

5 Conclusion and Future Direction

In this paper, we discussed how the computational time required to measure the accuracy loss due to approximation being introduced in CSs can be reduced by systematically selecting test samples that are likely to be more sensitive to the approximation, in order to provide a faithful yet efficient loss estimation. We provided an in-depth discussion concerning how approximation impacts CSs from an analytical standpoint, and we identified in the normalized Gini-Simpson index [21], or Gini impurity (9), the metric by which to discriminate those test samples whose predictions have higher sensitivity when approximation is introduced. We integrated our technique into two machine learning models to produce approximate variants of (CSs). Indeed, we employed current state-of-the-art methods in designing these models, incorporating various approximation techniques, classification tasks, datasets, and classifiers. We then assessed the accuracy loss using both exhaustive measurements with the entire test dataset and estimations from our approach. Our method demonstrated comparable performance while significantly reducing computational time.

In the next future, we intend to investigate how to take advantage of this approach for improving CS performance and to make CS less susceptible to adversarial attacks. In fact, by having the ability to measure how robust a given sample is with respect to classification, it is possible to field data model optimization procedures that focus on weakly robust samples.

References

[1] M. Barbareschi, S. Barone, and N. Mazzocca, "Advancing synthesis of decision tree-based multiple classifier systems: an approximate computing case study," *Knowledge and Information Systems*, pp. 1–20, Apr. 2021. [Online]. Available: https://link.springer.com/article/10.1007/s10115-021-01565-5

[2] M. Barbareschi, C. Papa, and C. Sansone, "Approximate Decision Tree-Based Multiple Classifier Systems," *International Conference on Optimization and Decision Science*, p. 47, Sep. 2017.

[3] V. Mrazek, Z. Vasicek, L. Sekanina, M. A. Hanif, and M. Shafique, "ALWANN: Automatic Layer-Wise Approximation of Deep Neural Network Accelerators without Retraining," *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, Nov. 2019, arXiv: 1907.07229. [Online]. Available: http://arxiv.org/abs/1907.07229

[4] Z.-G. Tasoulas, G. Zervakis, I. Anagnostopoulos, H. Amrouch, and J. Henkel, "Weight-Oriented Approximation for Energy-Efficient Neural Network Inference Accelerators," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4670–4683, Dec. 2020.

[5] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*. Austin, Texas: ACM Press, 2013, p. 1. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2463209.2488873

[6] A. Bosio, D. Ménard, and O. Sentieys, Eds., *Approximate Computing Techniques: From Component- to Application-Level*. Cham: Springer International Publishing, 2022. [Online]. Available: https://link.springer.com/10.1007/978-3-030-94705-7

[7] S. Barone, M. Traiola, M. Barbareschi, and A. Bosio, "Multi-Objective Application-Driven Approximate Design Method," *IEEE Access*, vol. 9, pp. 86 975–86 993, 2021, conference Name: IEEE Access.

[8] L. Sekanina, Z. Vasicek, and V. Mrazek, "Automated Search-Based Functional Approximation for Digital Circuits," in *Approximate Circuits*, S. Reda and M. Shafique, Eds. Cham: Springer International Publishing, 2019, pp. 175–203. [Online]. Available: http://link.springer.com/10.1007/978-3-319-99322-5_9

[9] E. Zacharelos, I. Nunziata, G. Saggese, A. G. Strollo, and E. Napoli, "Approximate recursive multipliers using low power building blocks," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1315–1330, 2022.

[10] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda, "Approximate Logic Synthesis: A Survey," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2195–2213, Dec. 2020, conference Name: Proceedings of the IEEE.

[11] M. Barbareschi, S. Barone, N. Mazzocca, and A. Moriconi, "A Catalog-based AIG-Rewriting Approach to the Design of Approximate Components," *IEEE Transactions on Emerging Topics in Computing*, 2022.

[12] M. Ahmadinejad and M. H. Moaiyeri, "Energy- and quality-efficient approximate multipliers for neural network and image processing applications," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 1105–1116, 2022.

[13] H. A. Almurib, T. N. Kumar, and F. Lombardi, "Approximate DCT Image Compression Using Inexact Computing," *IEEE Transactions on Computers*, vol. 67, no. 2, pp. 149–159, Feb. 2018.

[14] M. Barbareschi, S. Barone, A. Bosio, J. Han, and M. Traiola, "A Genetic-algorithm-based Approach to the Design of DCT Hardware Accelerators," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 18, no. 3, pp. 1–25, Jul. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3501772

[15] V. K. Chippa, D. Mohapatra, K. Roy, S. T. Chakradhar, and A. Raghunathan, "Scalable Effort Hardware Design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 9, pp. 2004–2016, Sep. 2014.

[16] S. Venkataramani, A. Raghunathan, J. Liu, and M. Shoaib, "Scalable-effort classifiers for energy-efficient machine learning," in *Proceedings of the 52nd Annual Design Automation Conference*. San Francisco California: ACM, Jun. 2015, pp. 1–6. [Online]. Available: https://dl.acm.org/doi/10.1145/2744769.2744904

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[18] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, and M. Martina, "An Updated Survey of Efficient Hardware Architectures for Accelerating Deep Convolutional Neural Networks," *Future Internet*, vol. 12, no. 7, p. 113, Jul. 2020, number: 7 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/1999-5903/12/7/113

[19] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the Accuracy and Hardware Efficiency of Neural Networks Using Approximate Multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, Feb. 2020.

[20] V. Mrazek, Z. Vasicek, L. Sekanina, H. Jiang, and J. Han, "Scalable Construction of Approximate Multipliers With Formally Guaranteed Worst Case Error," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 11, pp. 2572–2576, Nov. 2018.

[21] E. H. Simpson, "Measurement of Diversity," *Nature*, vol. 163, no. 4148, pp. 688–688, Apr. 1949, number: 4148 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/163688a0

[22] L. Breiman, J. H. Friedman, R. Olshen, and C. J. Stone, "Classification and Regression Trees," *Routledge*, 1984, publisher: Wadsworth.

[23] F. F. Claudio Stefano, "Avila," 2018. [Online]. Available: https://archive.ics.uci.edu/dataset/459

[24] UC Irvine Machine Learning Repository, "Dry Bean Dataset," 2020. [Online]. Available: https://archive.ics.uci.edu/dataset/602

[25] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, conference Name: Proceedings of the IEEE.

[26] Y. LeCun, C. Cortes, and C. Burges, "MNIST Handwritten digit database," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778. [Online]. Available: http://ieeexplore.ieee.org/document/7780459/

[28] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-10 (canadian institute for advanced research)," 2010. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[29] A. E. Eiben and S. K. Smit, "Evolutionary Algorithm Parameters and Methods to Tune Them," in *Autonomous Search*, Y. Hamadi, E. Monfroy, and F. Saubion, Eds. Berlin, Heidelberg: Springer, 2012, pp. 15–36. [Online]. Available: https://doi.org/10.1007/978-3-642-21434-9_2

[30] J. Blank and K. Deb, "A Running Performance Metric and Termination Criterion for Evaluating Evolutionary Multi- and Many-objective Optimization Algorithms," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. Glasgow, United Kingdom: IEEE, Jul. 2020, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/9185546/

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, and D. Cournapeau, "Scikit-learn: Machine Learning in Python," *MACHINE LEARNING IN PYTHON*, 2011.

[32] M. Koklu and I. A. Ozkan, "Multiclass classification of dry beans using computer vision and machine learning techniques," *Computers and Electronics in Agriculture*, vol. 174, p. 105507, Jul. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169919311573

[33] M. Hopkins, E. Reeber, G. Forman, and J. Suermondt, "Spambase Data Set," 1999. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/spambase

[34] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for large-scale machine learning," The Google Brain Team, Tech. Rep., 2015. [Online]. Available: https://www.tensorflow.org/

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," Dec. 2019, arXiv:1912.01703 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1912.01703

[36] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, Nov. 1999.

Mario Barbareschi is a Tenured Assistant Professor of Computer Systems at the Department of Electrical Engineering and Information Technologies of the University of Naples Federico II. He received the Ph.D. in Computer and Automation Engineering in 2015 from the University of Naples Federico II. His research interests include Hardware Security and Trust, Approximate Computing, emerging technologies, and embedded systems. He participates in international projects, collaborating with academic institutions and several industrial partners. He has authored more than 70 peer-reviewed papers published in leading journals and international conferences.

Salvatore Barone is an Assistant Professor at the Department of Electrical Engineering and Information Technology, University of Naples Federico II, Italy. He received his PhD in Information Technology and Electrical Engineering in 2022, from the University of Naples Federico II, Italy. His main research field is designing high-performance computing systems, even though his research interests include Safety Critical Systems, Railway Systems, and Embedded Systems based on the FPGA technology.