FPAX: A Fast Prior Knowledge-Based Framework for DSE in Approximate Configurations

Yuqin Dou, Chenghua Wang, Haroon Waris, Roger Woods, *Fellow, IEEE*, and Weiqiang Liu, *Senior Member, IEEE*

***Abstract*—Current artificial intelligence and data science applications typically require complex computations and massive amounts of data handling, presenting unprecedented challenges for embedded platforms. Approximate computing has emerged as the most promising design technique to address this issue, by providing a potential performance increase, while sacrificing accuracy within an acceptable range. Approximate arithmetic units require the creation of design space exploration techniques that can swiftly and automatically form an approximate configuration in fault-tolerant systems. Existing methods, however, use iterative design space sampling, resulting in a large amount of redundant computation. In this work, we propose the efficient FPAX automatic search framework which can learn from prior knowledge regarding the exploration process of known applications and use it to guide design exploration. This avoids excessive redundant computation and quickly provides an impressive approximate configuration. Compared with the jump search algorithm known for its efficiency, FPAX can also achieve faster convergence speed and better exploration quality. Even compared to our previous ENAP framework, it exhibits an 18× faster performance while achieving almost identical exploration quality for several commonly used fault-tolerant applications.**

***Index Terms*—Approximate computing (AC), approximate configuration, design space exploration (DSE), prior knowledge.**

I. Introduction

DATA handling aspects of evolving AI applications present huge computational issues for current hardware platforms. With the demise of Moore's law and Dennard scaling [1], technology advancement has slowed down and the aim of reducing computational power in recent high-density chips remains an unrivaled challenge. Approximate computing (AC) is a promising approach which has received attention [2], [3], [4], [5], [6], [7] as it improves hardware performance by relaxing accuracy metrics. It has been used in Google's tensor processing units (TPUs) to reduce power consumption and by IBM to accelerate AI in their chips [8].

Recent research has focused on the design of AC adders and multipliers, as these are commonly used in many applications which exhibit error-tolerant behavior [2], [3], [4], [5]. Their effective deployment depends on effective design space exploration (DSE) to produce higher quality configurations of AC units when replacing precise operations. The approximate configuration needs to satisfy user error constraints while minimizing metrics such as area and energy consumption. This can be cast as an optimization problem, f_{err}, such that

$$f_{\text{err}}\left(G, \text{App}_{\text{con}}\right) \leq \text{Err}_{\text{tar}} \tag{1}$$

$$\text{min.target}\left(G, \text{App}_{\text{con}}\right) \tag{2}$$

where G represents the application, App_{con} represents an approximate configuration, and Err_{tar}, the user error constraint. This optimization requires the replacement of exact operations with approximate units, at the various locations in the application and gives a large number of configurations with different properties, creating a huge search space. Therefore, a high-performance and automated search methodology is required.

Existing frameworks are mostly based on heuristic search methods, which sample the design space iteratively and then generate approximate configurations in each iteration. The approximate configuration with the smallest target metric is selected as the best. Although the random-based space sampling approach can obtain improved approximate configurations over multiple iterations, it leads to a large amount of redundant computation. Moreover, the exploration quality is ultimately limited by the parameter settings of the framework. Therefore, the improvement in the efficiency of DSE in approximate configurations is an open research problem.

In this work, we have traversed the similarities in the design exploration process for different, well-known applications, and created the FPAX framework (https://github.com/douyuqin/FPAX) to use these similarities to guide the design exploration process for new applications. It allows quick exploration of approximate configurations and does not rely on the existing general heuristic search methods. Utilizing learned knowledge, it avoids the redundant calculations used in previous DSE approaches, allowing rapid identification of high-quality approximate configurations with only minimal DSE effort. We show that it outperforms our recent heuristic search methodology (ENAP) [9], avoiding the multiple

Manuscript received 9 June 2023; revised 23 October 2023; accepted 17 December 2023. Date of publication 25 December 2023; date of current version 21 May 2024. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 92364201 and Grant 62022041. This article was recommended by Associate Editor X. Jiao. *(Corresponding author: Weiqiang Liu.)*

Yuqin Dou, Chenghua Wang, and Weiqiang Liu are with the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China (e-mail: douyuqin@nuaa.edu.cn; chwang@nuaa.edu.cn; liuweiqiang@nuaa.edu.cn).

Haroon Waris is with the Electrical Engineering Department, National University of Computer and Emerging Sciences, Islamabad 44000, Pakistan (e-mail: haroon.waris.v@nu.edu.pk).

Roger Woods is with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, BT9 5AH Belfast, U.K. (e-mail: r.woods@qub.ac.uk).

Digital Object Identifier 10.1109/TCAD.2023.3346289

iterations to obtain approximate configurations of similar quality. The major contributions of this article are summarized as follows.

1) To the best of our knowledge, FPAX is the first framework to utilize prior knowledge to guide DSE in approximate configurations, thereby greatly improving efficiency.
2) We propose the "Converter" methodology to transform the knowledge of FPAX into high-quality approximation configurations.
3) Compared with the jump search (JS) algorithm known for its efficiency, FPAX can obtain better results in less time. FPAX achieves an 18× speed improvement over our previous ENAP approach while having the same approximation configurations at the output.

This article is organized as follows. In Section II, we introduce AC and review existing DSE techniques used to implement it. The parameter definitions used in this work are presented in Section III. The motivating case study is then described in Section IV, and the FPAX framework is introduced in Section V. Section V gives details of the experiments undertaken using FPAX and compares the results with the latest work. Section VII provides the conclusions.

II. Background

AC is an emerging design technique that uses errors as a design dimension to permit a tradeoff between design accuracy and hardware resources. Two main methods are typically used: 1) voltage overscaling (VOS) and 2) design complexity reduction (DCR) [10], [11]. While VOS achieves energy savings through lower voltage, the errors introduced are nondeterministic due to the violation of timing constraints. Therefore, most research tends to focus on the DCR technique where the exact circuit is modified to obtain an approximate circuit, by adding/removing design elements. The Boolean logical expression of the exact circuit is modified, within the specified error limit, and as a result, an approximate circuit with less complexity is achieved.

In recent years, approximate arithmetic units have been extensively studied, focusing on approximate adders and multipliers. Lu [12] proposed the LUA approximate adder which is used to speed up the processor. Zhu et al. [13] proposed an error-tolerant adder, which uses exact calculations for the MSB bits and an XOR gate for LSB bits. An array multiplier design uses several different approximate compression units which are then used to create a digital FIR filter [14]. An open-source approximate unit library containing multiple approximate adders and multipliers with varying approximate levels, is presented in [15].

While approximate units have also been proposed for other functions, such as dividers [16], only approximate adders and approximate multipliers have been considered here. This is because as in other research [9], [17], [18], [19], they are the commonly used operations in a wide range of applications [20]. The mature development of these approximate arithmetic units presents both opportunities and challenges for exploring the design space of approximate circuits. The opportunity lies in the efficient automatic synthesis of fault-tolerant application systems, whereas the challenge is the ability to quickly obtain excellent approximate configurations within defined error constraints.

Mrazek et al. [21] have proposed a modified version of the stochastic hill climbing algorithm to select appropriate approximate units from the library to obtain the appropriate configuration. The method's search quality, though, is limited by the number of iterations. In [17], they use a random method to divide the design space into several promising regions and apply greedy methods to optimize the approximate configuration of each area. This method, however, mainly relies on the initial randomly distinguished regions which can affect the quality of the final result. Witschen et al. [22] used the JS algorithm to accelerate the exploration of design space, but this method easily falls into a local optimal solution. Due to the large design space of approximate configurations, researchers in [18] proposed to use error constraints to exclude high level approximate units, improving the exploration quality by pruning the search space.

In [9], we proposed the ENAP framework which utilizes the close relationship between the number of approximate units and error constraints to design a number-aware pruning technology. This paradigm further compresses the search space, using an improved genetic algorithm to complete the DSE. The method also randomly samples the design space after each iteration to improve the approximate configurations. Unfortunately, it requires a large computational budget to establish sufficient statistical data to provide a reasonable and good design. This article presents the FPAX framework to address this limitation, by utilizing prior knowledge to guide the design exploration of the approximate configuration.

Recent studies have shown how prior knowledge can be used to accelerate the DSE process. For example, Zhang et al. [23] proposed a DSE framework that combines prior knowledge with Bayesian optimization and can quickly find high-quality exploration results. Ferretti et al. [24] used initially learned knowledge from well-known Pareto dominance relationships among direct configurations. This knowledge is later applied to the new target, resulting in high-quality results with minimal synthesis runs. Machine learning can be used for evaluating the quality of HLS, but it requires a large number of samples for training. Kwon and Carloni [25] used transfer learning technology to obtain knowledge from the previously explored design space. Later, when exploring the new target design space, this method only required a small number of samples to obtain good evaluation quality.

III. Parameters Description

In this work, a significant number of different approximate level and error parameters, as defined in Table I, is used. In Fig. 1, the adder tree example requires approximation units from the library to create the approximate configuration. The units have already been arranged according to the AL_{au}, denoted by the numbers 1, 2, and 3, where higher levels indicate a greater amount of error and energy savings. In the approximate configuration shown in Fig. 1, nodes a_1 and

TABLE I
PARAMETERS DESCRIPTION

Parameter Names	Parameter Explanation
AL_{au}	Approximate level of a single approximate unit.
Err_{tar}	Error constraints from users.
Err_{exp}	Expected improvement in errors.
AL_{max}	Maximum approximate level that an application can achieve.
Err_{max}	Error of the approximate configuration at the maximum approximation level.
Err_{ac}	Error of the current approximate configuration.
AL_{ac}	Approximation level of the current approximate configuration.
AL_{exp}	Expected approximation level.
NAL_{ac}	Normalized approximation level in the current approximate configuration (NAL_{ac}=AL_{ac}/AL_{max}).
$NErr_{ac}$	Normalized approximation error in the current approximate configuration ($NErr_{ac}$=Err_{ac} /Err_{max}).

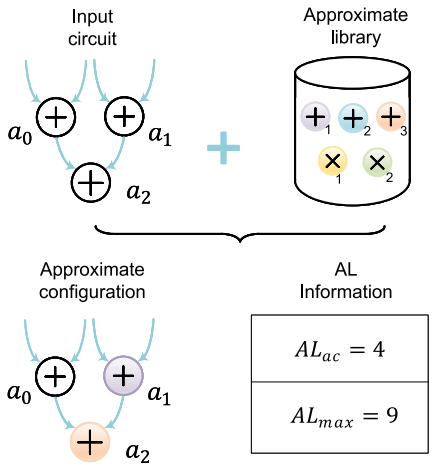


Fig. 1. Illustrative example to explain different approximate levels.

a_2 are replaced by approximate adders with $AL_{au} = 1$ and 3, giving an AL_{ac} of 4 for the layer. Since there are three configurable nodes in total, the $AL_{\max}$ is 9. NAL_{ac} is the normalized AL_{ac} of the approximate configuration. Hence, in this example, $\text{NAL}_{ac} = AL_{ac}/AL_{\max} = 4/9$.

IV. MOTIVATING CASE STUDY

This section aims to analyze the relationship between NAL_{ac}, the normalized approximation level of AL_{ac}, and $NErr_{ac}$, the normalized value of Err_{ac}, both for a given approximate configuration. This provides the initial motivation of this research with the various definitions provided in Table I. Fig. 2 depicts the exploration process of the approximate configurations for three applications, Sobel, Laplace, and 2×2 matrix multiplication, using the ENAP framework [9]. It presents the relationship between NErr_{ac} and NAL_{ac} where each point represents an approximate configuration.

ENAP can search for optimal approximate configurations by generating an approximate configuration on each iteration that is better than the previous version. From Fig. 2, we can observe a clear, proportional pattern between NAL_{ac} and NErr_{ac} when exploring optimal approximate configurations for different applications. As NAL_{ac} increases with NErr_{ac}, this implies greater energy savings.

Based on this observation, we propose the FPAX methodology to exploit this similarity in DSE process for the target application. For this, a preknown exploration process is defined as $F_{\text{pri}}(x)$, while the exploration process can be defined as $F_{\text{tar}}(x)$, which is unknown. Assuming an error constraint of Err_{tar}, $F_{\text{tar}}(\text{NErr}_{\text{tar}})$ is quickly obtained as $F_{\text{pri}}(x)$ and can be used to solve this problem due to the similarity of the exploration process. The similarity implies that while $F_{\text{pri}}(\text{NErr}_{\text{tar}})$ may not be equal to $F_{\text{tar}}(\text{NErr}_{\text{tar}})$, it may be very similar.[1] Hence, the relationship between $F_{\text{tar}}(\text{NErr}_{\text{tar}})$ and $F_{\text{pri}}(\text{NErr}_{\text{tar}})$ is described as

$$F_{\text{tar}}(\text{NErr}_{\text{tar}}) \geq \left|F_{\text{tar}}(\text{NErr}_{\text{tar}}) - F_{\text{pri}}(\text{NErr}_{\text{tar}})\right| \tag{3}$$

where NErr_{tar} is the normalized value of Err_{tar}. We note that a small error (NErr_{ac}) infers a small $F_{\text{pri}}(x)$ (NAL_{ac}) during the exploration process. In order to get closer to $F_{\text{tar}}(\text{NErr}_{\text{tar}})$, we can select a small error and combine $F_{\text{pri}}(x)$ with the previous $F_{\text{pri}}(\text{NErr}_{\text{tar}})$. In this way, we can utilize the similarity between $F_{\text{pri}}(x)$ and $F_{\text{tar}}(x)$ to approach the target in a guided manner. This negates the need for random sampling of the search space through multiple iterations as required in ENAP. We have successfully incorporated these benefits into the DSE to get approximate configurations. The proposed preliminary framework is illustrated in Fig. 3.

First, Err_{tar} needs to be normalized, and then fed into the function $F_{\text{pri}}(x)$, to provide a corresponding result, NAL_{ac}. Clearly, NAL_{ac} will not be equal to $F_{\text{tar}}(\text{NErr}_{\text{tar}})$, but will be similar. After obtaining NAL_{ac}, we convert it into an approximate configuration using the *Converter* function, allowing Err_{ac} to be obtained. Finally, we analyze if Err_{ac} meets the termination condition. If fulfilled, the program will exit and output the approximate configuration, otherwise it will update Err_{exp} and provide input to $F_{\text{pri}}(\text{NErr}_{\text{exp}})$ again, which can set a new NAL_{ac}. This NAL_{ac} value will then be combined with the previous approximate configuration and utilized by the *Converter* to form a new approximate configuration, corresponding to a new Err_{ac}, which will be analyzed again. The iterations will be carried out until we get obtain an error Err_{ac} that satisfies the termination condition.

In the preliminary proposed framework, $F_{\text{pri}}(x)$ represents prior knowledge that is formed before DSE and is used to

[1] $F_{\text{pri}}(\text{NErr}_{\text{tar}})$ may be slightly bigger than or less than $F_{\text{tar}}(\text{NErr}_{\text{tar}})$, but it will be closer to $F_{\text{tar}}(\text{NErr}_{\text{tar}})$ than initial value. The initial value refers to the $F_{\text{pri}}(0)$, which is equal to 0.

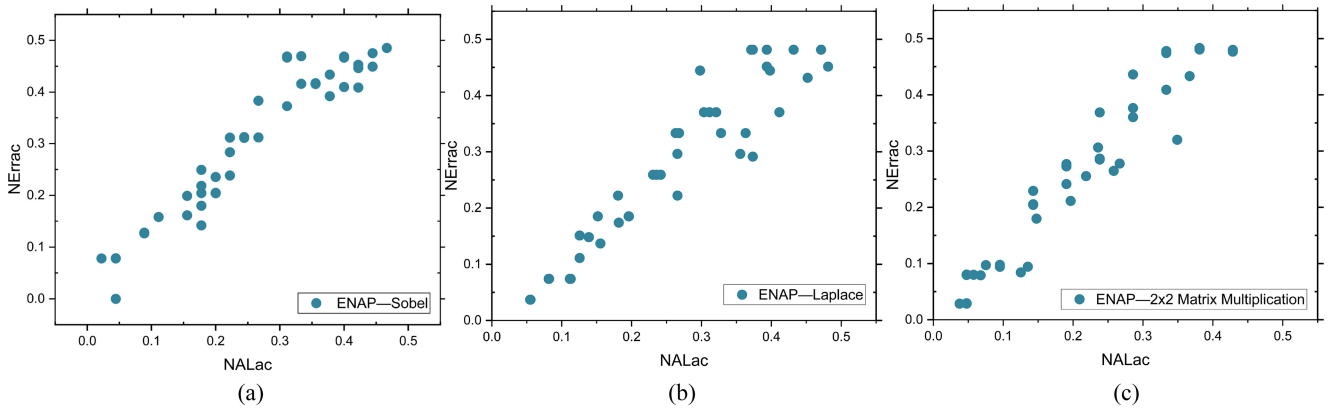


Fig. 2. Relationship between NAL_{ac} and NErr_{ac} in the exploration process for the (a) Sobel, (b) Laplace, and (c) 2 × 2 matrix multiplication operations.

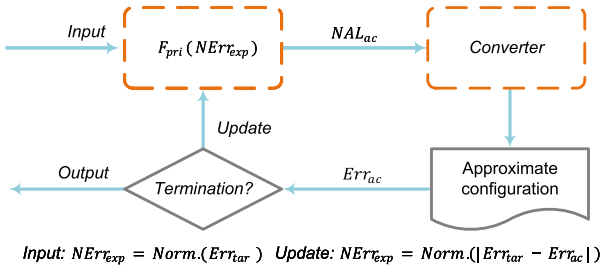


Fig. 3. Key FPAX optimization step.

guide the exploration of the target application, because of its similarity to $F_{\mathrm{tar}}(x)$. We expect that this will speed up the exploration process and will allow results close to $F_{\mathrm{pri}}(\mathrm{Err}_{\mathrm{tar}})$ to be quickly obtained. Fig. 3 indicates that the formation of $F_{\mathrm{pri}}(x)$ and the *Converter* are the key components and are thus detailed in the following sections, allowing the concept to be extended into a framework named FPAX.

V. Proposed FPAX Framework

The flow for the FPAX framework (Fig. 4) comprises: 1) prior knowledge acquisition and 2) DSE. Prior knowledge acquisition transforms a preknown exploration process into the function, $F_{\mathrm{pri}}(x)$, using a previously designed neural network (NN) model. Initially, ENAP is utilized to locate excellent approximate designs for a target application, and these are used to train the NN (as a predictor) which will be used for DSE.

In DSE, FPAX initially translates the high-level language of a target application into a data flow graph (DFG), and initializes the fundamental parameters. $\mathrm{Err}_{\mathrm{max}}$ refers to the error obtained when all nodes of the target application are replaced by the approximate units with the highest value of AL_{au}. $\mathrm{Err}_{\mathrm{max}}$ is used to calculate the normalized value of $\mathrm{Err}_{\mathrm{exp}}$, and is an one-time calculation effort.

$\mathrm{NErr}_{\mathrm{exp}}$ can be subsequently obtained and fed into the predictor, providing an NAL_{ac} budget which is used by the *Converter* to determine a promising approximate configuration. Afterward, this approximate configuration is analyzed with respect to the termination condition which if satisfied, terminates the process. Otherwise, $\mathrm{Err}_{\mathrm{exp}}$ is updated and fed into the predictor for further iteration.

A. Prior Knowledge Acquisition

The prior knowledge acquisition is conducted offline[2] and does not affect the efficacy of the DSE. Its objective is to transform an exploration process into an NN model, which can take the normalized value for expected improvement error ($\mathrm{NErr}_{\mathrm{exp}}$) as input and generate a suggested NAL_{ac} budget as defined in

$$\mathrm{NAL}_{ac} = F_{\mathrm{pri}}\left(\mathrm{NErr}_{\mathrm{exp}}\right). \tag{4}$$

The prior knowledge acquisition mainly consists of the following three parts.

1) Sample Generation: While the samples refer to the known approximate configurations, they are too large to use as training data, and our goal is to learn the exploration process. Therefore, there is no need to generate all possible approximate configurations. Instead, we employ ENAP to generate approximate configurations under different error constraints. Without loss of generality, the error constraints are set to $3, 5, 7, \ldots, \mathrm{Err}_{\mathrm{max}}/2$ where this latter constraint has been determined sufficient to generate enough samples. During the DSE in ENAP, all approximate configurations are saved.

2) Sample Selection: The approximate configurations obtained from *Sample Generation* are filtered and classified to enable the predictor $F_{\mathrm{pri}}(\mathrm{NErr}_{\mathrm{exp}})$ to easily converge. Identical approximate configurations are removed, and then the total AL_{ac} and Err_{ac} values are determined for each approximate configuration. Fig. 5 illustrates how all approximate configurations are classified according to their AL_{ac}. The first column represents the AL_{ac} and the second column displays all the Err_{ac} corresponding to it (Please note that all of the Err_{ac} values here are hypothetical). In order to ensure that the predictor $F_{\mathrm{pri}}(\mathrm{NErr}_{\mathrm{exp}})$ has a stable convergence (based on the reduction of errors), it is necessary to remove unsuitable approximate configurations. In Fig. 5,

[2]Offline refers to work that can be completed before DSE, it only needs to be run once. Hence, offline work does not impact the resource and time overhead of DSE. Online refers to the technology that runs during DSE, so online technology can affect the efficiency of DSE.

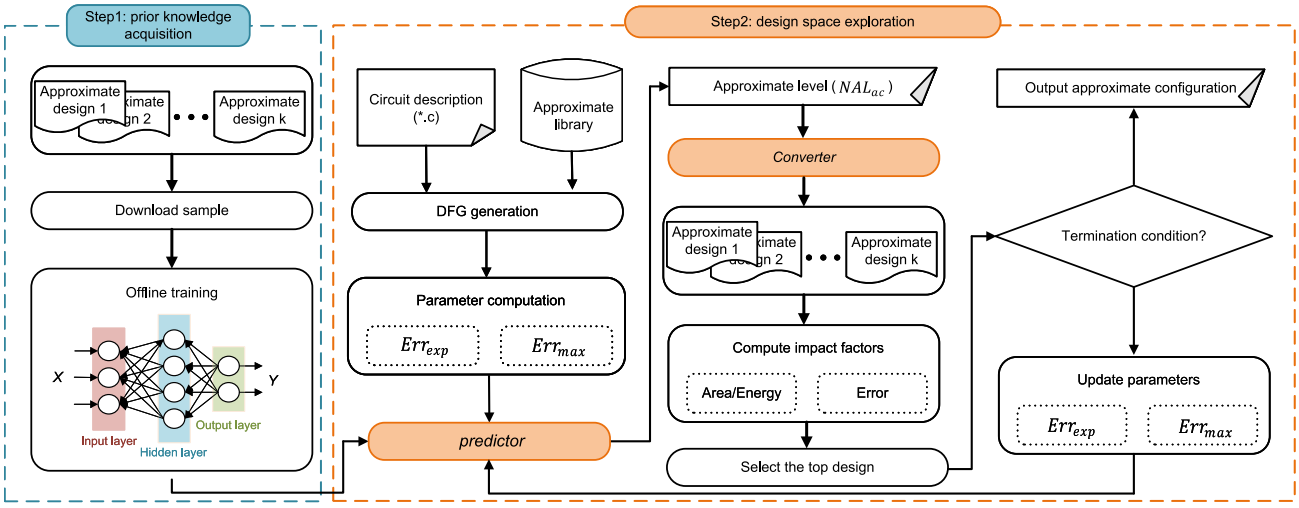


Fig. 4. Flow of FPAX.

there are two approximate configurations when $AL_{ac} = 1$, and the Err_{ac} are 1 and 2. However, when $AL_{ac} = 2$, the Err_{ac} of approximate configurations are 2 and 3. Therefore, we need to remove the approximate configuration with $\text{Err}_{ac} = 2$. Similarly, for $AL_{ac} = 3$, the approximate configuration with $\text{Err}_{ac} = 2$ needs to be removed. In other words, for each AL_{ac}, its corresponding Err_{ac} must be bigger than that of any smaller AL_{ac}. The filtered data ensures that the predicted values of $F_{\text{pri}}(\text{NErr}_{\text{exp}})$ converge as the error decreases. Fig. 5 also provides an example for displaying the data format of the input label and the output label. The last set of data after selection is treated as the example object, and from the example, we can know that both AL_{ac} and Err_{ac} need to be normalized. This is done by dividing the data by Err_{max} and AL_{max}, respectively, and then use it for the input label and output label of the NN model, respectively.

3) Prediction Models: The input data and the output labels obtained from above work are passed to train the NN model. The universal approximation theorem states that any continuous function defined on a compact set can be approximated by a fully connected NN [25]. Therefore, we have proposed a fully connected NN model as shown in Fig. 6. It consists of an input layer, four hidden layers, and an output layer where each small square represents a neuron, which is connected to a weight. These weights keep on changing until the NN model finishes the training on a provided input dataset. The input layer of the neural network takes the NErr_{exp} as input, and the layers of the network are connected by rectified linear unit (ReLU) functions. The proposed predictive function is shown in

$$F_{pri}(\text{NErr}_{\text{exp}}) = f_5^i\left(f_4^i\left(f_3^i\left(f_2^i\left(f_1^i(\text{NErr}_{\text{exp}})\right)\right)\right)\right) \tag{5}$$

where function $f_j^i(\text{NErr}_{\text{exp}})$ is defined as follows:

$$f_j^i(\text{NErr}_{\text{exp}}) = \text{ReLU}\left(\text{NErr}_{\text{exp}} * W_j^i + b_j^i\right) \tag{6}$$

where W_j^i is a weight matrix, and b_j^i is a bias vector. The learning objective is to minimize the difference between the actual NAL_{ac} and the predicted NAL_{ac}. The mean-squared error (MSE) is used to quantify the difference between the actual NAL_{ac} and the predicted NAL_{ac}, while the adaptive moment estimation (Adam) optimizer is used to improve the computational efficiency.

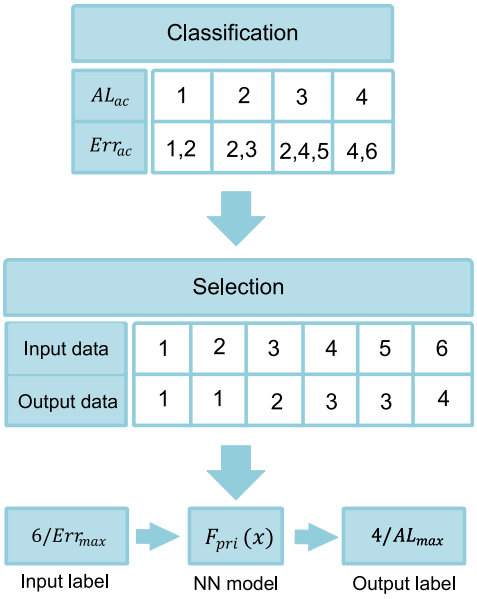


Fig. 5. Example for sample selection.

B. Converter

The output of Section V-A, i.e., the predictor, is a recommended NAL_{ac} budget, rather than an approximate configuration. The *Converter* creates an approximate configuration output, based on the cost-performance metric, defined as follows:

$$\text{Cost}_{\text{eff}} = \text{Save}_{\text{tar}}/\text{Err}_{ac} \tag{7}$$

where Save_{tar} represents the target savings and Err_{ac} represents the error of the given configuration, respectively. Clearly, to

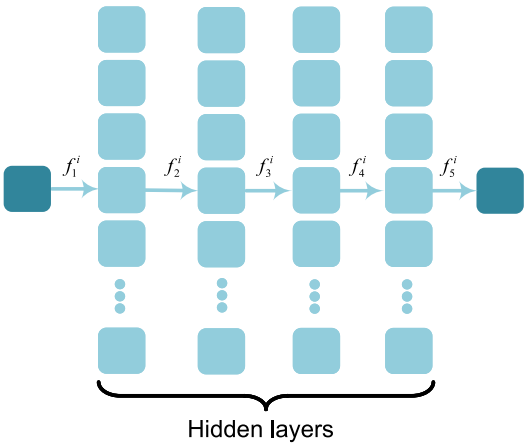


Fig. 6. NN model for learning prior knowledge.

Node	1	2	3	4	5	6
Operation	M	M	A	A	M	A
AL_{au}	1	1	2	3	3	4

Depth=6

Fig. 7. Unified form for approximate configuration.

achieve an excellent approximate configuration, a large Cost_{eff} value is desired. The concept of cost-effectiveness is used for the selection of approximate configurations in the *Converter*. Moreover, we have used the unified form of approximate configuration in DSE. The *Converter* requires approximate configurations in a uniform format, which should include information about adjacent nodes, approximation level and "Depth," an example of which is presented in Fig. 7.

In the six nodes in this example, *M* represents a multiplication, *A*, an addition and AL_{au} is the approximate level of each node. Each approximate configuration data form also requires "Depth," which refers to the location of the last changed node (for a given approximate configuration). As can be seen in Fig. 7, the sixth node is the last one to be changed for this approximation configuration. When a new approximate configuration is formed, FPAX begins to change nodes from the *Depth* node. The *Depth* parameter follows these guidelines.

1) *Depth* is equal to the location of the last changed node in the approximate configuration.
2) If *Depth* equals the location of the last node, it needs to be switched to the location of the node with the smallest AL_{au} (for the given approximate configuration). In Fig. 7, for example, *Depth* equals 6, which is the location of the last node. Therefore, to form a new approximate configuration, the FPAX starts improving from the first node and moves on to find the best candidate.

The purpose of the *Converter* is to convert NAL_{ac} into the best approximate configuration. It consists of the following steps.

Algorithm 1: Selection of the Approximate Unit

Input: DFG(G); Approximate Library (adder (A), multiplier (M)); *Depth*;
Output: The appropriate approximation units ($Cost_{ac}$);

1 **Parameter initialization**:
  $Cost_{ac} = [], flaga = 0, flagm = 0$
2 **for** *i in range(depth,length(G))* **do**
3 $(G[i], AL_{ac}) = Analysis(G(depth))$;
4 **if** *G[i]* $\in$ *'adder'& flaga==0* **then**
5 **for** *j in range(AL_{ac},length(A))* **do**
6 $G[i] \leftarrow A[j]$;
7 $(Err_{ac}, Save_{tar}) \leftarrow Com.factors(G[i])$;
8 $Cost_{ac} = Save_{tar}/Err_{ac}$;
9 $Cost_{ac}.append(A[j], Cost_{eff})$;
10 **end**
11 $flaga = 1$;
12 **end**
13 **if** *G[i]* $\in$ *'mul'& flagm==0* **then**
14 **for** *j in range(AL_{ac},length(M))* **do**
15 $G[i] \leftarrow M[j]$;
16 $(Err_{ac}, Save_{tar}) \leftarrow Com.factors(G[i])$;
17 $Cost_{ac} = Save_{tar}/Err_{ac}$;
18 $Cost_{ac}.append(M[j], Cost_{eff})$;
19 **end**
20 $flagm = 1$;
21 **end**
22 **end**
23 $Cost_{ac} = Rank(Cost_{ac})$;
24 Return $(Cost_{ac})$

1) Selection of the Approximate Units: Appropriate approximate units are selected for the *Converter*, based on cost-effectiveness, as shown in Algorithm 1. In Algorithm 1, *Com.factors* is the calculation function used to obtain parameters for approximate configuration (Err_{ac}, Save_{tar}), *flaga* is used to indicate that all approximate adders have been explored, and *flagm* is used to indicate that all approximate multipliers have been explored. The inputs are the DFG, Approximate Library (approximate adders and approximate multipliers), and the *Depth*. It reads the DFG, then starts the analysis from *Depth* node. The analysis includes the type of current node and the AL_{ac} (lines 2 and 3 in Algorithm 1) of the current node. Algorithm 1 then replaces the current node with an approximate unit with a bigger AL_{ac} value, calculates the cost-effectiveness of the replaced approximate configuration, and saves the approximate unit and the cost-effectiveness details. The process continues by replacing the current node with an approximate unit with a bigger AL_{ac} value and saving the approximate unit and its corresponding cost-effectiveness until all configurable approximation units have been replaced (steps 5–9 and 14–19 in Algorithm 1).

Once the cost-effectiveness of all configurable approximate units is acquired, these approximate units are arranged in descending order (with reference to the cost-effectiveness parameter). The first two approximate units with the highest cost-effectiveness are retained and used for further processing.

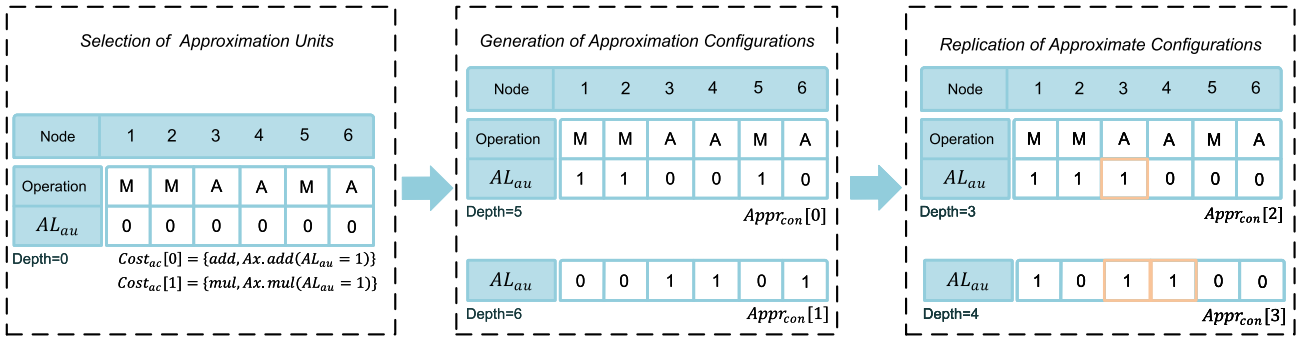


Fig. 8. Example for replication of approximate configurations.

2) Generation of Approximate Configurations: The appropriate approximate units obtained from *Selection of the approximate unit* are combined with the AL_{ac} (provided by the predictor) to form an approximate configuration. This methodology (Algorithm 2) requires a DFG, appropriate approximate units (Cost_{ac}), NAL_{ac}, and *Depth* as inputs. Initially, it analyzes the *Depth* node of DFG to determine whether it can be replaced by $\text{Cost}_{ac}[0]$, which needs to have the same operation type and a smaller AL_{au}. If the condition is fulfilled the node is replaced with $\text{Cost}_{ac}[0]$; otherwise, the next node of the target application is determined that satisfies the condition.

It is evident that replacing the node with $\text{Cost}_{ac}[0]$ increases AL_{ac}. Therefore, it is important to determine if the increased AL_{ac} exceeds the NAL_{ac} by normalizing it and then comparing with NAL_{ac}. If AL_{ac} is under the range, then replacement is carried out; if it exceeds or equals the NAL_{ac}, then Algorithm 2 is stopped. During the node replacement process, if the accumulated AL_{ac} has not reached NAL_{ac}, then $\text{Cost}_{\text{ac}}[1]$ will continue to complete this task until both approximate units are replaced or NAL_{ac} is satisfied. To effectively handle the node replacement process, Algorithm 2 is run twice. The first preferred replacement approximation unit is $\text{Cost}_{ac}[0]$, and the second is $\text{Cost}_{ac}[1]$. The two approximate configurations are formed after two runs, which are used for processing in the subsequent steps.

Please note that *Generation of Approximate Configurations* is distinct from the classical *graph traverse + greedy*. Classical *graph traverse + greedy* entails a one-time traversal of all nodes with a fixed starting point. In contrast, our method does not restrict itself to a single node traversal; if the error constraint is not exceeded, our method continues to traverse nodes. Furthermore, the starting point for our method's traversal is determined by both the *Depth* and the node with the lowest level approximation unit.

Algorithm 2: Generation of Approximate Configurations

Input: DFG(G); NAL_{ac}; $Cost_{ac}$; *Depth*;
Output: The optimal approximation configurations ($Appr_{con}[]$);

Parameter initialization: $flag = 0$;
$AL_{\text{exp}} = NAL_{ac} * AL_{\max}$;
for *q in range(0,2)* **do**
  for *i in range(Depth,length(G))* **do**
    if $jungle(G(i), Cost_{ac}[0]/[1]) == 1$ **then**
      $G[i] \leftarrow Cost_{ac}[0]/[1]$;
      $flag \leftarrow flag + AL_{au}$;
      $Depth = i$;
      if $flag \geq AL_{\text{exp}}$ **then**
        $Appr_{con}.append(G)$;
        break
      end
    end
  end
  for *i in range(Depth,length(G))* **do**
    if $jungle(G(i), Cost_{ac}[1]/[0]) == 1$ **then**
      $G[i] \leftarrow Cost_{ac}[1]/[0]$;
      $flag \leftarrow flag + AL_{au}$;
      $Depth = i$;
      if $flag \geq AL_{\text{exp}}$ **then**
        $Appr_{con}.append(G)$;
        break
      end
    end
  end
end
Return ($Appr_{con}$); *Depth*

3) Replication and Selection of Approximate Configurations: This step realizes more approximate configurations from those received from *Generation of Approximate Configurations*. Moreover, a systematic selection procedure is adopted to finalize the most cost-effective approximate configuration among those available. Fig. 8 is used as an example to describe the replication method.

In this example, the initial input configuration is an exact configuration, and the two suitable approximate units are determined by the Selection of Approximation Units (which are approximate multiplication and approximate adder with $AL_{au} = 1$). After that, two approximate configurations ($\text{Appr}_{\text{con}}[0]$ and $\text{Appr}_{con}[1]$) are formed according to the budget of $\text{NAL}_{ac} = 3$.

During replication, the first approximated node in $\text{Appr}_{\text{con}}[1]$ is replicated onto the corresponding node in $\text{Appr}_{\text{con}}[0]$. In each replication, only approximate nodes changed by this iteration are considered, e.g., relative to the

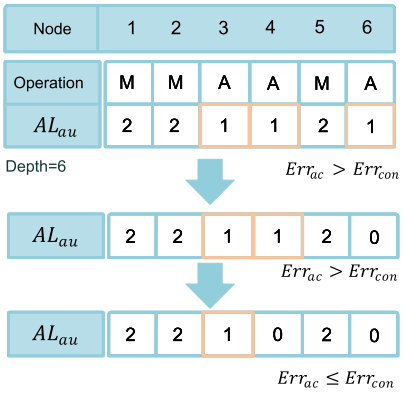


Fig. 9. Example for selection of approximate configurations.

initial configuration, the first approximated node in $\text{Appr}_{\text{con}}[1]$ is node 3. Therefore, the configuration information of node 3 in $\text{Appr}_{\text{con}}[1]$ is replicated onto node 3 in $\text{Appr}_{\text{con}}[0]$), and finally to meet the $\text{NAL}_{ac} = 3$ budget, the last approximated node of $\text{Appr}_{\text{con}}[0]$ is reverted to an exact node, which conforms to $\text{Appr}_{\text{con}}[2]$. The first two approximated nodes of $\text{Appr}_{\text{con}}[1]$, are required to be replicated onto the corresponding nodes in $\text{Appr}_{\text{con}}[0]$ (the configuration information of node 3 and node 4 in $\text{Appr}_{\text{con}}[1]$ are replicated onto the node 3 and node 4 in $\text{Appr}_{\text{con}}[0]$).

To fulfill the range of $\text{NAL}_{ac} = 3$, the last two approximated nodes in $\text{Appr}_{\text{con}}[0]$ are reverted to exact nodes, so that the $\text{Appr}_{\text{con}}[3]$ is constructed. Similarly, we consider the first three approximate nodes in $\text{Appr}_{\text{con}}[1]$, followed by the first four, and so on, until all the approximate nodes in $\text{Appr}_{\text{con}}[1]$ are replicated (in this example, there are only three approximate nodes in $\text{Appr}_{\text{con}}[1]$). The final step of this replication process selects the top approximate configuration based on the cost-effectiveness characteristic.

C. Termination Conditions

The proposed FPAX needs to determine the termination conditions before starting DSE. It searches for the best approximate configurations through multiple iterations, with each iteration producing a configuration with a bigger Err_{ac} than the previous iteration. When FPAX obtains an approximate configuration that exceeds the Err_{tar}, it produces the final output. Fig. 9 demonstrates how the Err_{ac} of the approximate configuration is assumed to exceed the Err_{tar}, and the approximate nodes in the yellow box are assumed to be approximate nodes that are changed in the last iteration. Therefore, if AL_{au} of approximate nodes in the yellow box are all equal to 0, then the Err_{ac} of the approximate configuration is smaller than Err_{tar}. Moreover, the AL_{ac} of the best approximate configuration must be between the approximate configuration of the previous and current iteration.

In order to achieve the best approximation configuration, therefore, AL_{au} in the yellow box is sequentially reduced, only reducing one approximation node at a time with AL_{au} by 1. Simultaneously, Err_{ac} is calculated until the Err_{tar} condition is satisfied, after which the reduction is stopped. For example, in Fig. 9, initially, the AL_{au} of node 6 is reduced followed by the calculation of Err_{ac}. If Err_{ac} satisfies the Err_{tar}, then the reduction of AL_{au} of node 5 is undertaken until the Err_{tar} condition is fulfilled. Eventually, the approximate configuration obtained is regarded as the final output.

D. Design Space Exploration

Before introducing DSE, we also need to introduce a new variable, *thr* which is a threshold used by FPAX to achieve a better approximate configuration. NAL_{ac} is a normalized value that needs to be multiplied by AL_{max} and then rounded to obtain AL_{exp}. However, when the NAL_{ac} is relatively small, AL_{exp} could be equal to 1 as the replicated approximation configuration might be the same as the original configuration, and it will not generate a new configuration. In that particular scenario, FPAX can only use very few approximate units, and the replication of the approximation configuration becomes invalid, limiting the diversity of approximation configurations. Therefore, we define a new parameter *thr* to ensure that the AL_{exp} is at least equal to 2 for each iteration, as this is equal to Err_{au} of the second approximated unit in the list sorted by errors in descending order.

It is pertinent to mention that the second smallest error is selected as the value of *thr* once the Err_{au} is sorted in ascending order. This is mainly used to help FPAX achieve a better approximate configuration. NAL_{ac} is a normalized value that needs to be multiplied by AL_{max} and then rounded to obtain AL_{exp}. However, when the NAL_{ac} is relatively small and the AL_{exp} could be equal to 1, FPAX can only use very few approximate units, and the replication of the approximation configuration will become invalid. This is because AL_{exp} also needs to be considered during replication; When $AL_{\text{exp}} = 1$, i.e., the replicated approximation configuration is the same as the original one, then it will not generate a new configuration. We believe that it will affect the diversity of approximation configurations, so we have defined a threshold to ensure that the AL_{exp} is at least equal to 2 for each iteration. The value of *thr* is defined as equal to Err_{au} of the second approximated unit in the list sorted by errors in descending order - by applying each approximated unit one at a time and sorting the resulting Err_{au} in ascending order, the second smallest error is selected as the value of *thr*.

The proposed DSE, presented in Algorithm 3, takes the DFG, target error Err_{tar}, and *thr* as inputs. The DSE first transforms Err_{tar} into Err_{exp} and then compares Err_{exp} with *thr*. If $\text{Err}_{\text{exp}} <= thr$, then it sets $AL_{\text{exp}} = 2$. If $\text{Err}_{\text{exp}} > thr$, it is necessary to feed Err_{exp} to the predictor, and then the *Converter* will use the results of the predictor and the current approximate configuration to realize an improved approximate configuration. Subsequently, the Err_{ac} of the approximate configuration is calculated and compared with the Err_{tar}. If the result is less than the Err_{tar}, the approximate configuration is saved and the next iteration is started. However, if the Err_{ac} exceeds the Err_{tar}, then the termination process is performed and the approximate configuration obtained is considered as the final output.

Algorithm 3: DSE

Input: DFG(G); Err_{tar}; thr;
Output: The excellent approximation configuration;
Parameter initialization: $flag = 0$; $Err_{\text{exp}} = 0$; $Err_{ac} = 0$;$Err_{\max}$;$AL_{\max}$
$current.Appr_{con} = G$ **for** i *in range(0, 100)* **do**
  $Err_{\text{exp}} = Err_{con} - Err_{ac}$;
  if $Err_{\text{exp}} <= thr$ **then**
    $AL_{\text{exp}} = 2$
  else
    $NErr_{ac} = Err_{\text{exp}}/Err_{\max}$;
    $NAL_{ac} = F_{pri}(NErr_{ac})$;
    $AL_{\text{exp}} = NAL_{ac} \times AL_{\max}$;
  end
  $App_{con} = Converter(AL_{\text{exp}}, current.Appr_{con})$;
  $Err_{ac} = f(App_{con})$;
  if $Err_{ac} > Err_{con}$ **then**
    $Final.Appr_{con} =$ $Termin.process(Curr.App_{con}textreplacement)$;
    break
  else
    $current.Appr_{con} = App_{con}$
  end
end
Return ($Final.Appr_{con}$);

TABLE II
APPLICATIONS AND THEIR CHARACTERISTICS

Application program	Nodes	Add/Multiply	Area ((um^2))	Energy (μW)
3×3 Convolution	17	8 / 9	7516.8	4095
FIR filter	25	12 / 13	10925.2	5947
Laplace filter	9	7 / 1	1834.2	967
Sobel	15	9 / 4	7243.1	2356

The area and energy of applications were measured by Synopsys Design Compiler in 45nm NanGate technology [15]

VI. EXPERIMENT

Four well-known fault-tolerant applications (Table II) from signal and image processing, are selected to evaluate the FPAX. Table II contains the number of nodes, operation types, area, and energy consumption for each application. All applications are encoded in C and parsed into DFG using GAUT [26]. The approximate units are extracted from [15] and have their corresponding areas and energy consumption mentioned in Table III. It is worth noting that FPAX can be applied to any approximate unit with area and energy consumption values. mean error distance (MED) and peak signal-to-noise ratio (PSNR) are used to evaluate the fault tolerance of approximate circuits, which are defined in [9]. MED and PSNR are selected as accuracy metrics for this experiment. FPAX employs error estimation methods like both [9] and [22], all of which rely on simulation-based approaches, thus, FPAX provides the quality of the approximate configuration and supports any accuracy metric. 100 000 random samples are used to calculate the Err_{ac} of approximate circuits.

TABLE III
AREA AND POWER CONSUMPTION FIGURES FOR APPROXIMATE LIBRARY UNITS

Approximate Level	Resource Name	Energy (μW)	Area (um^2)
Approximate multiplier			
AL_{au} = 1	mul8_EXZ	380	663.0
AL_{au} = 2	mul8_150Q	360	660.3
AL_{au} = 3	mul8_CK5	345	604.5
Approximate adder			
AL_{au} = 1	add16u_0EM	57	115
AL_{au} = 2	add16u_08F	52	106.1
AL_{au} = 3	add16u_1JH	51	144.4

Due to different test benchmarks or different optimization targets, it is not appropriate to compare our approach with all frameworks. Therefore, for a fair comparison, the ENAP in [9] and the JS algorithm suggested in [22], which are consistent with our optimization targets, are used to compare with FPAX. ENAP is the latest exploration framework for approximate configuration and the JS algorithm is a highly efficient heuristic search algorithm renowned for its ability to rapidly accomplish DSE.

FPAX and the proposed algorithms were implemented in Python and experiments were performed on a computer with a 3.10-GHz Intel Core I5-10500 CPU with 6 cores and 16-GB RAM. FPAX is evaluated against the recently published ENAP in terms of quality and efficiency using the same approximation units given in Table III. We set up the model, $F_{\text{pri}}(\text{NErr}_{\text{exp}})$ in Fig. 6, with four hidden layers, each corresponding to nodes 8, 8, 16, and 4. The exploration process of 2 × 2 matrix multiplication under ENAP is used to train it. In this experiment, given the adoption of two accuracy metrics, it necessitates the use of two prediction models, corresponding to two sets of samples. One set is derived from approximate configurations based on MED, while the other set is derived from approximate configurations based on PSNR, both originating from ENAP. ENAP generated a total of 2000 approximate configurations for each metric. After applying our selection method, the dataset for MED retained 932 samples, while the dataset for PSNR retained 873 samples. Consistent with ENAP, FPAX uses its method [19] to calculate energy consumption and area.

A. Exploration Quality

The search ability of heuristic algorithms is closely related to parameter settings. In this experiment, the setting of ENAP parameters is consistent with [9], and the design of JS algorithm in [22]. Figs. 10 and 11 detail the results of the three methods for the minimum energy and area under different error constraints, respectively. The y-axis gives the normalized area and power consumption and the x-axis presents error constraints. It is important to note that the smaller the normalized value is, then the better the quality of exploration. The figures show that as the error constraints increase, the area and energy consumption for both decrease. In most cases, FPAX exhibits the lowest normalized area/energy, while ENAP and JS exhibit comparable performance in different scenarios. This suggests that FPAX demonstrates superior exploration quality.

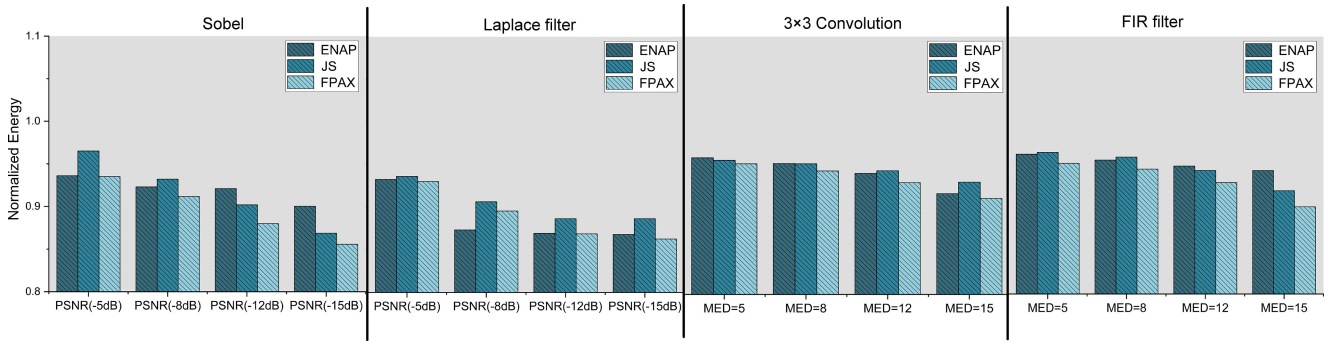


Fig. 10. Energy saving results for different benchmarks against various error bounds.

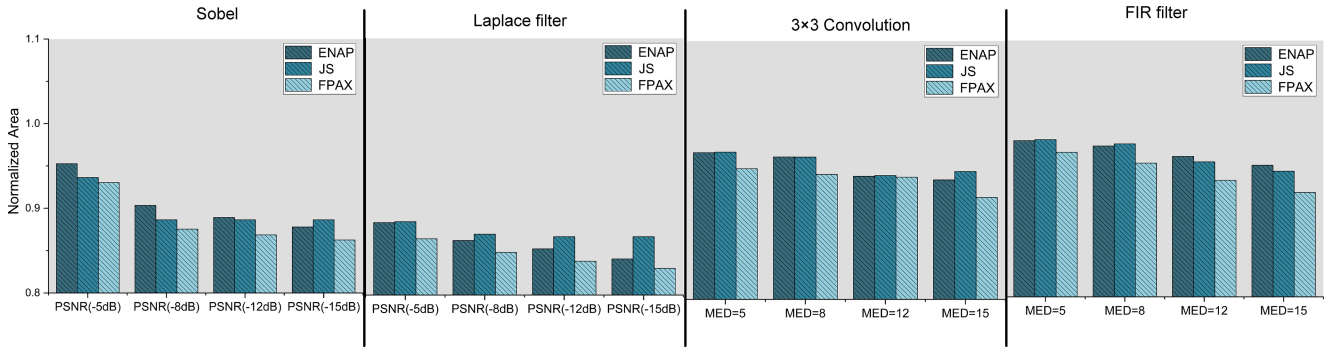


Fig. 11. Area saving results for different benchmarks against various error bounds.

In some scenarios, ENAP also generates better results compared to FPAX. For example, when the error constraints of Laplace are small, ENAP exhibits a slightly better exploration quality than FPAX as it has a reduced search space of Laplace under small error constraints, allowing it to approach the optimal approximation configuration with fewer iterations. However, in general, FPAX shows better exploration quality because FPAX features prior knowledge to approach the best approximation configurations. From Fig. 12, the exploration quality of JS is shown to be better than ENAP in some cases, such as for FIR filter for the MED = 15 error bound. This is mainly because according to [9], the number of iterations of ENAP is set to 3. We release the iteration limit of ENAP in the next section for more in-depth analysis.

B. Time Overhead

As FPAX, ENAP, and JS are different approaches, the number of iterations cannot be used to compare the efficiency, so the time consumed is used as shown in Fig. 12. It reflects the change in the exploratory quality of approximate configurations as reflected in normalized energy consumption, over time. ENAP uses a genetic algorithm-based search framework, whose search quality converges to an optimal approximation configuration as the number of iterations increases.

To allow comparison, the ENAP termination condition is modified to an infinite number of iterations until the same result occurs three consecutive times. The termination condition of JS is consistent with [22], that is, when an approximate configuration that satisfies the error constraint first occurs, the result is output. In Fig. 12, the red line, purple line, and blue line correspond to the exploration processes of FPAX, JS, and ENAP, respectively, where each symbol represents an iteration. The exploration quality and time overhead at the end of iteration for these three methods are presented. Clearly, from the perspective of exploration quality, in most cases, FPAX can achieve the best exploration quality, followed by ENAP and JS. From the perspective of time efficiency, FPAX also requires the least time, followed by JS, and ENAP requires the longest time. From Fig. 12, it can be observed that FPAX terminates the first iteration of ENAP when the error bounds are MED = 5 and MED = 8. This means that FPAX gives the results before the end of the first iteration of ENAP. In addition, the quality of FPAX exploration is much better than that of the first iteration of ENAP.

In the most obvious example, when the error bound applied to 3 × 3 convolution is MED = 5, the quality of exploration at the end of ENAP iteration is worse than that at the end of FPAX iteration. Meanwhile, the time consumption at the end of ENAP iteration is 595.411 s, and that at the end of FPAX iteration is only 32.467 s. This means that FPAX achieves the best approximation configuration 18× faster than ENAP.

For some cases, ENAP exhibits a slightly better quality of exploration than FPAX before the end of iteration, such as when the error bound of 3 × 3 convolution is MED=15. However, this is achieved at the cost of large time overhead. ENAP requires 638.134 s to obtain similar results to FPAX, which is about 6× more than 99.912 s of FPAX. In summary, FPAX uses the least time cost among the three methods, and achieves better or identical quality exploration results when compared with ENAP.

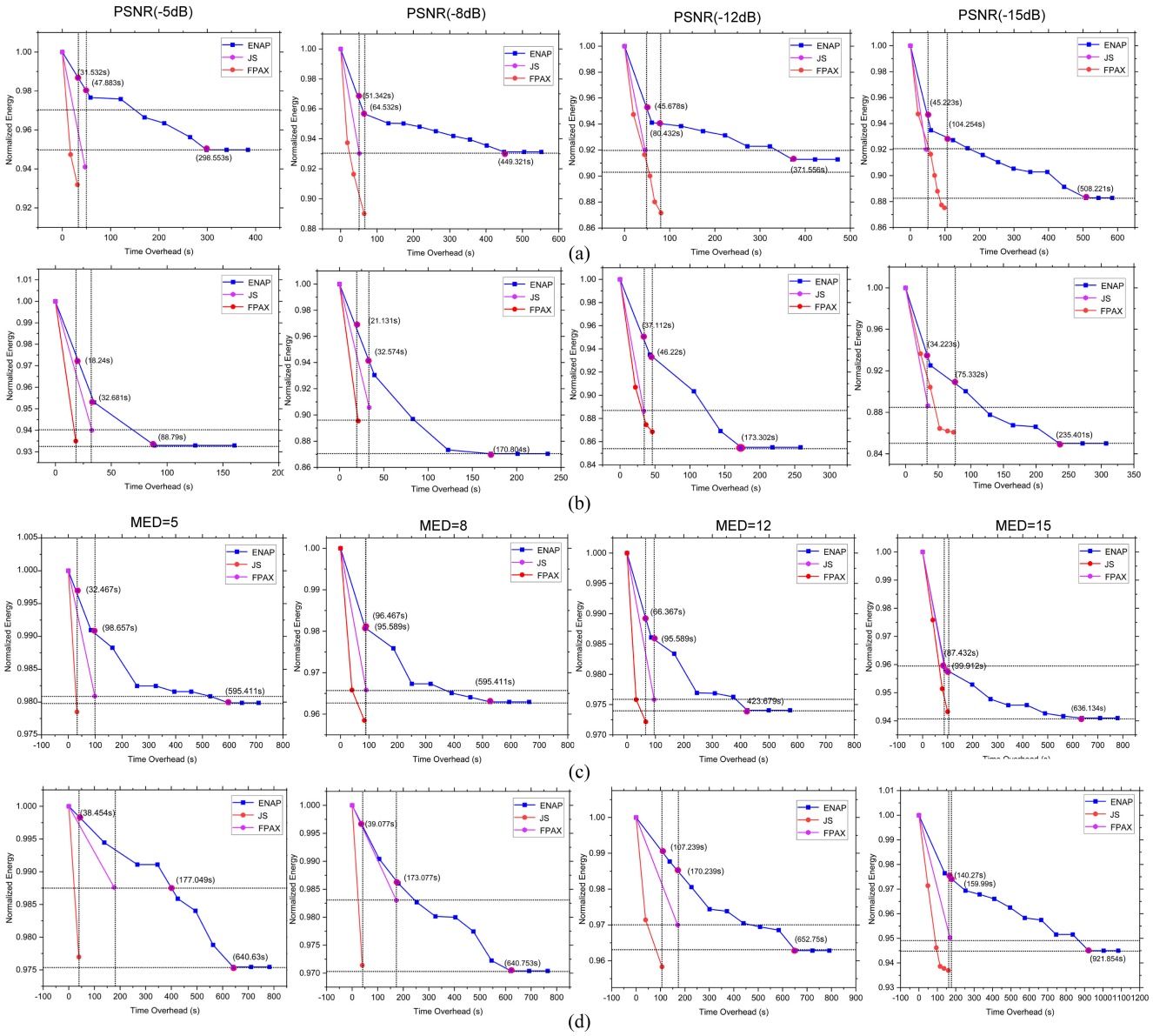


Fig. 12. Comparison of exploration efficiency between ENAP, JS, and FPAX. (a) Sobel. (b) Laplace filter. (c) 3×3 convolution. (d) FIR filter.

TABLE IV
COMPARISON OF TIME OVERHEAD USING ENAP AND FPAX IN DIFFERENT APPLICATIONS

search method	Average single time (s) ×number of iterations				application
	PSNR(-5dB)	PSNR(-8dB)	PSNR(-12dB)	PSNR(-15dB)	
FPAX	15.76×2	17.114×3	16.05×5	17.37×6	Sobel filter
ENAP	50.12×8	50.133×11	53.43×9	50.926×10	
FPAX	18.24×1	21.131×1	15.40×3	15.06×5	Laplace filter
ENAP	44.93×5	42.701×5	43.35×6	39.17×8	
	MED=5	MED=8	MED=12	MED=15	
FPAX	32.487×1	48.2335×1	33.18×2	30.1004×3	3×3 Convolution
ENAP	74.42×10	75.11×10	70.611×8	79.525×10	
FPAX	16.64×1	24.69×2	15.766×5	18.05×6	Fir filter
ENAP	91.5×9	81.21×10	82.85×10	91.24×12	

Table IV shows the average single iteration time and number of iterations required for FPAX and ENAP. From Table IV, it can be found that the average single iteration time of FPAX is smaller than that of ENAP, especially in applications with larger search space (such as FIR filter). This is mainly because these applications with large search space contain multiple operations, and estimating the error quality of an approximate configuration requires more time. FPAX can avoid a large number of redundant calculations, that is, evaluating approximate configurations less frequently than ENAP, thus saving time and resources. Table IV also shows that the number of iterations of FPAX is always smaller than that of ENAP, which once again proves that the convergence speed of FPAX is faster than that of ENAP.

VII. CONCLUSION

A new framework, called FPAX, has been proposed to create AC designs for fault-tolerant applications, giving faster performance and almost identical exploration quality than previous work. FPAX first learns knowledge from well-known

explorations offline, and then applies the learned knowledge to DSE in approximate configuration, to avoid a large amount of redundant calculations. As learned knowledge is not directly approximate configurations, we also propose a method called *Converter* that can transform this knowledge into high-quality approximate configurations. Compared with the JS algorithm known for its efficiency, FPAX can also achieve faster convergence speed and better exploration quality. When compared with the latest search framework ENAP, FPAX can achieve similar exploration quality 18 times faster than ENAP.

REFERENCES

[1] M. Foster and N. Forbes, "Guest editors introduction: The end of Moore's law?" *Comput. Sci. Eng.*, vol. 5, no. 1, pp. 18–19, Jan. 2003.

[2] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, Jan. 2013.

[3] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf.*, 2015, pp. 1–6.

[4] H. Waris, C. Wang, W. Liu, and F. Lombardi, "AxBMs: Approximate radix-8 booth multipliers for high-performance FPGA-based accelerators," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 5, pp. 1566–1570, May 2021.

[5] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 393–401, Feb. 2017.

[6] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. 18th IEEE Eur. Test Symp.*, 2013, pp. 1–6.

[7] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited: Cross-layer approximate computing: From logic to architectures," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf.*, 2016, pp. 1–6.

[8] W. Liu, F. Lombardi, and M. Shulte, "A retrospective and prospective view of approximate computing," *Proc. IEEE*, vol. 108, no. 3, pp. 394–399, Mar. 2020.

[9] Y. Dou, C. Wang, R. Woods, and W. Liu, "ENAP: An efficient number-aware pruning framework for design space exploration of approximate configurations," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 5, pp. 2062–2073, May 2023.

[10] Y. Dou, C. Gu, C. Wang, W. Liu, and F. Lombardi, "Security and approximation: Vulnerabilities in approximation-aware testing," *IEEE Trans. Emerg. Topics Comput.*, vol. 11, no. 1, pp. 265–271, Jan.–Mar. 2023.

[11] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, p. 62, Mar. 2016. [Online]. Available: https://doi.org/10.1145/2893356

[12] S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, vol. 37, no. 3, pp. 67–73, Mar. 2004.

[13] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 8, pp. 1225–1229, Aug. 2010.

[14] L.-D. Van, S.-S. Wang, and W.-S. Feng, "Design of the lower error fixed-width multiplier and its application," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 47, no. 10, pp. 1112–1118, Oct. 2000.

[15] V. Mrázek, R. Hrbáček, Z. Vašíček, and L. Sekanina, "EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. Design, Autom. Test Europe Conf. Exhibit.*, 2017, pp. 258–261. [Online]. Available: https://www.fit.vut.cz/research/publication/11262

[16] S. Hashemi, R. I. Bahar, and S. Reda, "A low-power dynamic divider for approximate applications," in *Proc. 53rd Annu. Design Autom. Conf.*, 2016, pp. 1–6.

[17] M. Vaeztourshizi, M. Kamal, and M. Pedram, "EGAN: A framework for exploring the accuracy vs. energy efficiency trade-off in hardware implementation of error resilient applications," in *Proc. Int. Symp. Qual. Electron. Design*, 2020, pp. 438–443.

[18] D. Ma, R. Thapa, X. Wang, X. Jiao, and C. Hao, "Workload-aware approximate computing configuration," in *Proc. Design, Autom. Test Europe Conf. Exhibit.*, 2021, pp. 920–925.

[19] J. Castro-Godínez, J. Mateus-Vargas, M. Shafique, and J. Henkel, "AxHLS: Design space exploration and high-level synthesis of approximate accelerators using approximate functional units and analytical models," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design*, 2020, pp. 1–9.

[20] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *Proc. Conf. Design, Autom. Test Europe*, 2008, pp. 1250–1255.

[21] V. Mrazek, M. A. Hanif, Z. Vasicek, L. Sekanina, and M. Shafique, "AutoAx: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

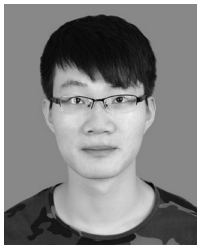
[22] L. Witschen, H. G. Mohammadi, M. Artmann, and M. Platzner, "Jump search: A fast technique for the synthesis of approximate circuits," in *Proc. Great Lakes Symp. VLSI*, 2019, pp. 153–158.

[23] Z. Zhang, T. Chen, J. Huang, and M. Zhang, "A fast parameter tuning framework via transfer learning and multi-objective Bayesian optimization," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, 2022, pp. 133–138.

[24] L. Ferretti, J. Kwon, G. Ansaloni, G. Di Guglielmo, L. P. Carloni, and L. Pozzi, "Leveraging prior knowledge for effective design-space exploration in high-level synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3736–3747, Nov. 2020.

[25] J. Kwon and L. P. Carloni, "Transfer learning for design-space exploration with high-level synthesis," in *Proc. ACM/IEEE Workshop Mach. Learn.*, 2020, pp. 163–168.

[26] E. Martin, O. Sentieys, H. Dubois, and J. L. Philippe, "GAUT: An architectural synthesis tool for dedicated signal processors," in *Proc. Eur. Design Autom. Conf.*, 1993, pp. 14–19.



Yuqin Dou received the B.S. degree in electrical engineering and automation from the Xi'an University of Technological Information, Xi'an, China, in 2016, and the M.S. degree in information engineering from Xi'an Technological University, Xi'an, in 2019. He is currently pursuing the Ph.D. degree in electrical and information engineering with the Nanjing University of Aeronautics and Astronautics, Nanjing, China.

His research interests mainly include hardware security and electronic design automation.

Chenghua Wang received the B.Sc. and M.Sc. degrees from Southeast University, Nanjing, China, in 1984 and 1987, respectively.

In 1987, he joined the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, where he became a Full Professor in 2001. He has published six books and over 100 technical papers in journals and conference proceedings. His current research interests include testing of integrated circuits and systems for communications.

Mr. Wang is the recipient of more than ten teaching and research awards at the provincial and ministerial level.

Haroon Waris received the Ph.D. degree in communication and information engineering from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in March 2022.

His Ph.D. research work was accepted in Ph.D. forums of DAC 2021, ASP-DAC 2021, ETS 2021, and ISVLSI 2022. He was invited to CASS Mentoring Program@ISCAS2021 (only ten mentees have been selected). He has published one book chapter by Springer House, two patents, and over 15 journal and conference papers. His research interest mainly focuses on approximate computing, hardware security, VLSI design for DSP, and analog/digital IC design and verification.

Dr. Waris serves as an External Reviewer for IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS, IEEE TRANSACTIONS ON EMERGING TOPIC IN COMPUTING, IEEE TRANSACTIONS ON COMPUTERS, and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: EXPRESS BRIEFS. He is the PC member of IEEE Asian Hardware Oriented Security and Trust Symposium (AsianHOST 2022/23).

Roger Woods (Fellow, IEEE) received the B.Sc. and Ph.D. degrees from Queen's University Belfast, Belfast, U.K., in 1985 and 1990, respectively.

He is a Professor of Digital Systems and the Dean of Research with the Faculty of Engineering and Physical Sciences, Queen's University Belfast. He co-founded the spin-off company, Analytics Engines Ltd., Belfast, where he acts as the Chief Scientist. His research interests are in heterogeneous programmable systems for data analytics and embedded systems for medical and smart city applications.

Prof. Woods was elected as a Royal Academy of Engineering Fellow in acknowledgment of his contributions to entrepreneurship and innovation. He is the Industry and Exhibition Chair of ISCAS2025 and is a member of the Industrial Electronics and Signal Processing societies.

Weiqiang Liu (Senior Member, IEEE) received the B.Sc. degree in information engineering from the Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China, in 2006, and the Ph.D. degree in electronic engineering from the Queen's University Belfast, Belfast, U.K., in 2012.

He is currently a Professor and the Vice Dean of the College of Electronic and Information Engineering and the College of Integrated Circuits, NUAA. He has published two research books and over 200 leading journal and conference papers (over 90 IEEE and ACM journals, including nine invited papers). His research interest focuses on energy efficient and secure computing integrated circuits and systems.

Prof. Liu has been awarded the prestigious Excellent Young Scholar Award by National Natural Science Foundation of China in 2020 and the Young Scientist Award by Fok Ying Tung Education Foundation, Ministry of Education, China, 2022. He has been listed in the Stanford University's 2020 list of the top 2% scientists in the world. He is the Vice President for Technical Activities of the IEEE Nanotechnology Council (NTC). He serves as the Steering Committee Chair for IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS, IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, and IEEE TRANSACTIONS ON COMPUTERS, and a Guest Editor for PROCEEDINGS OF THE IEEE. He is the Program Co-Chair of IEEE ARITH 2020, ACM/IEEE NANOARCH 2022, and IEEE AsianHOST 2023. He is a Tutorial Organizer and a Speaker in DAC 2022, DATE 2022, IEEE ISCAS 2021, and COINS 2021. He is a member of IEEE NTC AdCom and CASCOM/VSA Technical Committee of IEEE CAS Society.