# Workload-Aware Approximate Computing Configuration

Dongning Ma[‡], Rahul Thapa[‡], Xingjian Wang[‡], Cong Hao[§], Xun Jiao[‡],
[‡]Villanova University, [§]Georgia Institute of Technology

***Abstract*—Approximate computing recently arises due to its success in many error-tolerant applications such as multimedia applications. Various approximation methods have demonstrated the effectiveness of relaxing precision requirements in a specific arithmetic unit. This provides a basis for exploring simultaneous use of multiple approximate units to improve efficiency. In this paper, we aim to identify a proper approximation configuration of approximate units in a program to minimize energy consumption while meeting quality constraints. To do this, we formulate a constrained optimization problem and develop a tool called `WOAxC` that uses genetic algorithm to solve this problem. `WOAxC` considers the impact of different input workload on the application quality. We evaluate the efficacy of `WOAxC` in minimizing the energy consumption of several image processing applications with varying size (i.e., number of operations), workload (i.e., input datasets), and quality constraints. Our evaluation shows that the configuration provided by `WOAxC` for a system with multiple approximate units improves the energy efficiency by, on average, 79.6%, 77.4%, and 70.94% for quality loss of 5%, 2.5% and 0% (no loss), respectively. To the best of our knowledge, `WOAxC` is the first workload-aware approach to identify proper approximation configuration for energy minimization under quality guarantee.**

## I. Introduction

Approximate computing has recently arisen as a promising solution for improving energy efficiency in post-Moore's era. Its success has been demonstrated in many modern applications such as image processing [11], [4], machine learning [3], [8], and bio-signal analysis [14]. These applications typically show inherent error tolerance, i.e., they do not require completely accurate computations for delivering acceptable output quality. Utilizing such error tolerance, approximate computing "intentionally" seeks to design imperfect hardware to trade off accuracy for better energy efficiency. Specifically, researchers have developed approximate arithmetic units through inexact logic structure such as approximate adders [2], [11], [7] and multipliers [16], [10], [5]. While offering improvements, these methods also face several challenges to put approximate computing to practical use.

The first challenge is how to control the output quality, especially considering that output quality of approximate computing are heavily application and input dependent [6], [3]. The second challenge is how to identify a proper configuration of approximate computing from a large search space. This is because approximate units usually have multiple approximation settings. For example, the approximate adder developed in [2] has four settings, each of which has an unique structure, resulting in different energy-error profile. Using a uniform configuration for all the target units across the entire program may limit the potential gains and calls into question the value of approximate computing.

There are several studies on this problem [3], [9], [15]. For example, Gupta *et al.* proposed a gradient descent-based optimization method to select approximation settings for machine learning applications [3] in an hardware-independent manner, and Hu *et al.* proposed integer linear programming (ILP)-based optimization to select approximate settings [9] in FPGA. In this work, we will also focus on hardware-independent approximation configuration at the program level. However, in these studies, the final output quality is typically formulated as a mathematical function of the approximation error of individual approximate units. For example, the final output in [9] is expressed as a linear function of each individual approximate unit. Such formulation does not fully reflect the impact of input workload on quality estimation.

To tackle such limitations, we propose a workload-aware algorithmic approach that can automatically identify proper approximation configuration for energy minimization under quality guarantee. This is implemented in a tool called **`WOAxC`**. We formulate the problem as a constrained combinatorial optimization problem and solve it using evolutionary searching algorithm. We perform three steps to construct **`WOAxC`**: (1) integrating approximate units into the data flow graph (DFG) of the given program and emulating the program over the given input workload, (2) formulating the problem of minimizing energy under quality constraints as a constrained combinatorial optimization problem, and (3) identifying the proper approximation configuration using an evolutionary search algorithm to solve the optimization problem.

Specifically, our contributions are as follows:

- We propose **`WOAxC`**, to the best of our knowledge, the first workload-aware approximate computing configuration tool that can automatically identify proper approximation settings for energy minimization with quality guarantee.
- To enable an efficient search, **`WOAxC`** employs two phases, pruning and searching. The pruning phase can significantly compress the solution space, after which the searching phase uses an evolutionary algorithm to search the proper solution.
- We evaluate the efficacy of **`WOAxC`** across three image processing applications, three input datasets, and three quality constraints. Our evaluation shows that **`WOAxC`** can improve the energy consumption by, on average, 79.6%,

77.4%, and 70.94% for quality loss of 5%, 2.5% and 0% (no loss), respectively.

## II. Related Work

Approximate computing recently emerges as a promising computing paradigm for many error-tolerant applications such as multimedia and machine learning. Many studies focused on developing new approximate hardware units with flexible configurations. For example, in [2], authors proposed a reconfigurable approximate carry look-ahead adder (RAP-CLA) that could switch between approximate and exact operating modes. In [16], authors proposed AQ-LETAM, an output quality-tunable multiplier that allowed the users to define the level of approximation based on input characteristics or application quality constraints. Xu et al. proposed a runtime reconfigurable manager to select the proper approximate design based on a presumed input data distribution [18]. This reconfigurability leads to a tradeoff between efficiency and error at circuit level.

On the other hand, several studies were proposed to enable a system-level tuning of approximate units under an end-to-end quality constraints [3], [9], [15], [13]. LEMAX [3] used gradient descent-based method to optimize approximation settings of different approximate units for machine learning applications. Hu *et al.* optimized the approximation settings of adders and multipliers using an ILP-based optimization method [9]. autoAx [13] selects the most suitable approximate circuits from available libraries to generate an approximate accelerator for a given application. It construct a machine learning model to predict the output quality largely largely based on the hardware configuration without considering different input workload. Achilles [15] selects between three different quality management modes based on a quality predictor. A common limitation for all these studies is that they did not consider the impact of different input workload on the output quality. Actually, even under same approximation configuration, different input workload will lead to different output quality.

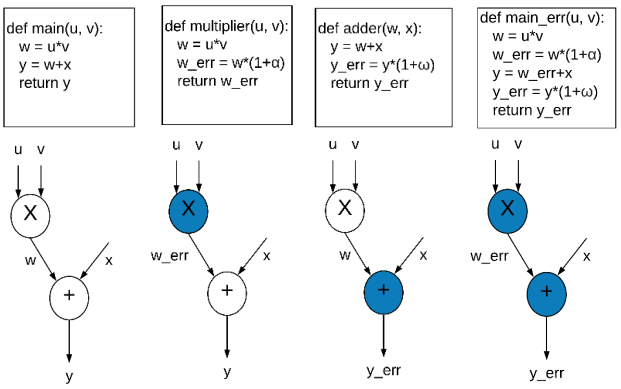
Unlike these previous studies, **WOAxC** presents the first effort to identify the proper approximation configuration by incorporating the impact of input workload. **WOAxC** identifies the approximation configuration on a per-application and per-dataset basis because the experimental results show that different applications/datasets have different amenability to approximation. Further, instead of employing a mathematical model for evaluating the output quality [9], [3], **WOAxC** performs real program DFG emulation with error injection to obtain the output quality.

## III. Motivating Case Study

This section describes how errors are injected to programs, and the impact of input workload on approximate computing.

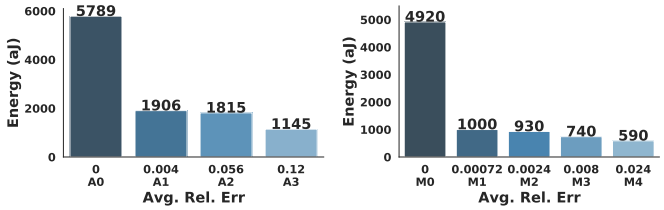
### A. Error Injection to Program DFG

We target two most-widely used approximate units, approximate adders and approximate multipliers. We choose approximate adders from RAP-CLA [2], which includes four




(a) Original (b) Apx. multiplier (c) Apx. adder (d) Apx. adder and multiplier

Fig. 1. The injection of approximation error into data flow graph. $\omega$ and $\alpha$ are the average relative error of adder and multiplier respectively.



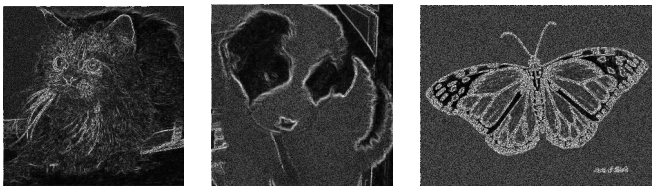

(a) Approximate Adders (b) Approximate Multipliers

Fig. 2. Energy-error trade-off for approximate adders and multipliers under different settings. A0 and M0 are the exact setting, and A$X$ and M$X$ are approximation settings.

approximation configurations (referred as $A0$ - $A4$). We choose approximate multipliers from AQ-LETAM [16], which includes five approximation configurations (referred as $M0$ - $M5$). We use the relative error as the error metric to represent the approximation error of each unit, based on which we inject errors to the program DFG. The error injection of approximate multipliers and adders to DFGs are presented in Fig. 1. Take an adder as an example, if the original addition is $c = a+b$, the post-approximation result would become $c = (a+b)*(1+\omega)$, where $\omega$ is the relative error. We illustrate the energy-error tradeoff of approximate units with different approximation settings in Fig. 2, based on which we can see that the energy decreases with the increase of approximation error. Note that while we focus on relative error as error metric in this paper, **WOAxC** can be naturally extended to other error metrics such as error rate because the quality evaluation is performed by emulating real programs.

### B. Impact of Input Workload

We use Sobel filter as an example, one of the most widely-used image processing application, to show the impact of input workload on the quality of approximate computing. The DFG of Sobel filter has 18 multipliers and 17 adders, each of which can adopt a different setting.



(a) Cat 22.71dB (b) Dog 16.83dB (c) Butterfly 13.36dB

Fig. 3. Same approximate setting applied to Cat, Dog, and Butterfly images for Sobel filter

We randomly choose an approximation configuration for all the 35 target units. Then, we feed three different images as input for Sobel filter — cat, dog, and butterfly. As shown in Fig. 3, the output of the three input images are different. The output of cat image has a PSNR of 22dB while the output of dog and butterfly images have noted distortions with only 16dB and 13dB respectively. This difference suggests that even under same approximation configuration, different input workload will lead to different output quality. This key observation motivates us to explore workload-aware approximate computing.

## IV. WOAxC Framework

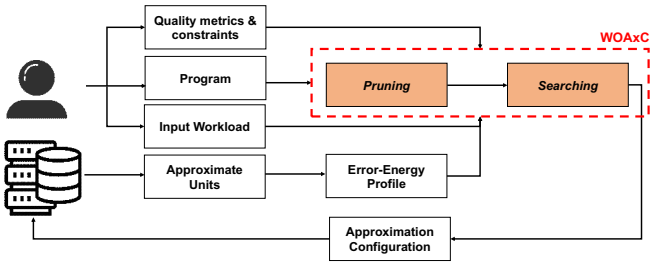



Fig. 4. WOAxC framework

The identification of a proper approximation setting can be formulated as a constrained combinatorial optimization problem, i.e., find an approximation configuration vector $s$ for all the target units in an application that:

$$\min_{s} Energy(A, I, s) \quad s.t.\ QualityLoss(A, I, s) < \sigma \tag{1}$$

where $A$ represents the application program, $I$ represents the input workload and $\sigma$ is the user-defined constraint of quality loss. Note that the quality loss is a strong function of $I$.

The entire flow of the **WOAxC** is illustrated in Fig. 4, with several key steps: first, users will feed the information regarding application program $A$, input workload $I$ and quality constraints $\sigma$ into the **WOAxC**. The error-energy profile of approximate units will also be fed into **WOAxC**. **WOAxC** will then determine a specific configuration for the available configurable approximate units to minimize energy consumption, while meeting the quality constraints. The most intuitive way to find the solution is through brute-force search. However, the search space complexity is prohibitively huge: if there are $n$ computation units and each unit has $m$ different configurations, the search space size is $m^n$. Using Sobel filter as an example, the search space is $5^{18} * 4^{17} = 6.5536e{+}22$. Directly applying heuristic solution will result in a significantly long search time.

To tackle such issue, we develop a two-phase approach for **WOAxC**: pruning and searching. Pruning phase will first reduce the search space by eliminating infeasible solutions. This is key to enabling an efficient search action. Then, the search phase will use evolutionary algorithm to search for the solution.

### A. Pruning

**Algorithm 1** **WOAxC** Pruning

**Input** The application $A$; list of possible settings $P$.
**Output** Feasible search space $S$

1: $S = \{\}$
2: **for** each unit $A[i]$ in application $A$ **do**
3: $A[i] \leftarrow P[0]$ //$P[0]$ is the unit with minimum error in $P$.
4: **end for**
5: **for** each unit $A[i]$ in application $A$ **do**
6: **for** each available setting $P[j]$ in $P$ **do**
7: $A[i] \leftarrow P[j]$ and run application $A$
8: **if** $QualityLoss(A) > QualityConstraint()$ **then**
9: $S.append(P[j-1])$
10: $A[i] \leftarrow P[0]$
11: break
12: **end if**
13: **end for**
14: **end for**
15: **return** $S$

The pruning process is shown in Algorithm 1. We start with examining if each individual unit is amenable to approximation, and, if so, to what extent. When examining the approximation amenability of a specific unit, all the other units are configured without any approximation, i.e., using exact settings $A0/M0$ (Line 2, Line 3). For this specific unit, the approximation level will be sequentially enhanced and applied, starting from no approximation ($A0/M0$), to $A1/M1$, $A2/M2$, etc.

Under each approximation setting, the program DFG will be emulated under corresponding error injection and the output quality will be evaluated against the user-defined quality constraints. If the output satisfies the user-defined constraints, the approximation level will be further enhanced. Then, the program DFG will be emulated and the output quality will be evaluated again. Similarly, this process will be iteratively executed until the output quality does not meet the constraints (Line 8). This suggests that previous approximation setting is the boundary approximation level for this unit and cannot be enhanced further, otherwise the quality constraint will be violated.

By repeating such process for each target unit in the DFG, we can obtain a boundary setting for each unit (Line 9). Any setting beyond this boundary set will be infeasible due to quality constraints violation. In the rare case that if the lowest approximation level of a certain unit can lead to a quality

violation, this unit is deemed as a "critical" unit and is not amenable to approximation at all. The output of this pruning process eliminates a significant amount of infeasible solutions, which drastically reduces the search space of applications.

### B. Searching

**Algorithm 2** `WOAxC` Searching

**Input** The application $A$; feasible search space $S$.
**Output** Optimized population $C$

```
Randomly generate initial population C within S
for each chromosome C[k] in C do
    run A with settings of C[k]
    if QualityLoss(A) > QualityConstraint() then
        delete C[k] from C
    end if
end for
while not terminate do
    C.children ← GA(C) //Generate children based on the current population
    C ← TopFit(C, C.children) //Only the fittest can survive
end while
return C
```

After obtaining the pruned solution space, we develop a customized search process based on genetic algorithm (GA), one of the most-widely used evolutionary algorithms, to solve this combinatorial optimization problem. The basic idea of GA is that natural evolution will choose the fittest species over time. Note that many heuristic methods such as simulated annealing can be used to solve the problem. While the selection and comparison of different heuristic methods are beyond the scope of this paper, GA naturally fits our problem because the approximation settings can be easily interpreted and encoded as genes.

To customize and apply genetic algorithm in finding the solution, we define the following items:

- Definition - representation of chromosomes
- Population generation - crossover and mutation
- Population selection and elimination - fitness function
- Termination conditions

We describe our customized GA in Algorithm 2, which consists of the following stages.

***Chromosomes*** We define each individual chromosome as an approximation configuration vector. Each gene in the chromosome indicates the approximation configuration for a specific unit. The length of a chromosome depends upon the number of target units in a given application.

***Population*** After pruning, a set of initial populations is randomly generated. Note that, every individual in this initial population must meet quality constraints. If one individual fails to meet the constraints, it will be immediately eliminated.

In `WOAxC`, we apply two genetic operations to generate future generations, crossover and mutation. Crossover is to produce a child chromosome with segments from two parent chromosomes based on one (or more) crossover points. We adopt random two-point crossover mechanism which randomly selects two crossover points in the chromosome. The genes between the two crossover points from the two chromosomes will be swapped to produce children.

Mutation is to randomly modify the gene on one parent chromosome to produce a child chromosome. We adopt random two-point mutation algorithm which randomly selects two mutation points in the chromosome and replace them with random genes.

***Fitness Function*** The fitness function is to determine which individuals can survive. Since our objective is to minimize the energy consumption, we compute the fitness value based on the energy consumption of each individual. The fitness function is based on elitism selection, i.e., only a certain number of individuals with the best fitness values will survive. Empirically, we retain the best 10 individuals in our experiment.

***Termination Condition*** Termination conditions indicate the end of searching. If the GA iteration count or running time surpasses the user-defined limitation, or if further generations cannot produce more elite children, searching will terminate. After this step, we can have solutions to the optimization problem.

## V. Experimental Results

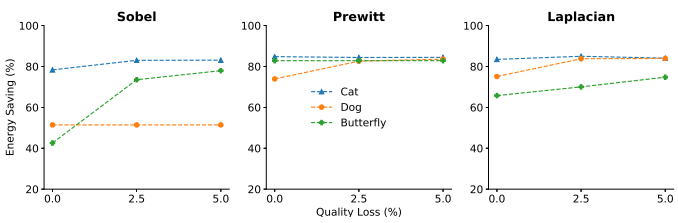



Fig. 5. Energy saving under different quality loss

### A. Experimental Setup

We select and implement three widely-used image processing applications: Sobel Filter, Prewitt Filter and Laplacian Filter. Both Sobel filter and Prewitt filter have 35 target units, among which 18 units are multipliers and the other are adders. Laplacian filter has 17 units, among which 9 are multipliers and the other are adders. We adopt three different datasets: Butterlfy Dataset [17], Dogs dataset, and Cats Dataset [1]. We implement `WOAxC` in Python. We define $PSNR \geq 30dB$ as acceptable quality for a single image [12]. For a dataset, we define the quality loss $q$ as $q = \frac{n}{m}$, where $n$ is the number of unacceptable images (i.e., PSNR<30dB), while $m$ is the total number of images in the dataset. Without loss of generality, we assume three different quality constraints ($q$): 0%, 2.5%, and 5%.

***Baseline*** A direct comparison with existing approaches are infeasible [9], [15], [13], [3] because their problem formulation is not so tightly connected with input workload: we measure the quality loss through a direct program DFG emulation over input workload and approximation configuration, while the existing approaches mostly estimate quality loss based on approximation configuration only. Therefore, we compare

TABLE I
SEARCH SPACE REDUCTION BY PRUNING

| | Sobel | | | Prewitt | | | Laplacian | | |
|---|---|---|---|---|---|---|---|---|---|
| **Search Space Size** | butterfly | cat | dog | butterfly | cat | dog | butterfly | cat | dog |
| Before Pruning | 6.55E+22 | 6.55E+22 | 6.55E+22 | 6.55E+22 | 6.55E+22 | 6.55E+22 | 1.28E+11 | 1.28E+11 | 1.28E+11 |
| After Pruning | 2.62E+10 | 9.06E+14 | 3.69E+9 | 1.97E+11 | 1.64E+17 | 8.96E+12 | 1.01E+5 | 3.28E+7 | 2.025E+5 |
| Compression ratio ($\approx$) | 2E+12 | 7E+7 | 1E+13 | 3E+11 | 4E+5 | 7E+9 | 13E+5 | 39E+2 | 63E+4 |

**WOAxC** with a workload-unaware approach that uses GA to identify an approximation configuration under one dataset but also use it for other datasets. For example, the approximation configuration identified under butterfly dataset will be used for dog dataset.

### B. Energy Saving

TABLE II
AVERAGE ENERGY SAVING.

| **Quality loss (%)** | Sobel | Prewitt | Laplacian |
|---|---|---|---|
| 5.0 | 74.15 | 83.63 | 80.92 |
| 2.5 | 69.31 | 83.28 | 79.59 |
| 0.0 | 57.44 | 80.52 | 74.86 |

We first present the results of pruning for the applications in Table I, where we can see that the search space is compressed by 3900X to $10^{13}X$. Then, using **WOAxC**, we identify the approximation configuration for each application and compute the energy saving results under different quality loss, as shown in Fig. 5 and Table II.

Even under most stringent 0% quality loss constraint, **WOAxC** is able to save significant energy. **WOAxC** can enable 57.44%, 80.52%, and 74.86% energy saving on average across three datasets on Sobel filter, Prewitt filter, and Laplacian filter respectively. Under 5% quality constraint, **WOAxC** can enable 74.15%, 83.63%, and 80.92% energy saving on Sobel filter, Prewitt filter, and Laplacian filter respectively. For a specific application, e.g., Sobel filter, **WOAxC** saves energy consumption at 42.57% (butterfly), 78.35% (cat), and 51.41% (dog). This also applies to other applications and datasets.

Further, different applications have different amenability to approximation and this amenability can change with different quality constraints. For example, under absolute zero quality loss, Prewitt filter shows the highest approximation level with an average energy saving of 83.63% and Sobel filter is worst at 57.44%. Different datasets of the same application also result in different approximation configurations and energy saving. For example, for Sobel filter, cat dataset always results in the highest energy saving. This is because, as verified by our experiments, cat dataset is the most error tolerant. This further motivates an workload-aware approximation configuration. Actually, this phenomena also applies to the other two applications.

Generally, as the quality constraints are relaxed, the energy saving also increases. This is expected because more relaxed quality constraint means that higher approximation level can be applied. However, there are also exceptions depends on the datasets and applications. For example, in Sobel filter, as the quality constraints relax, dog dataset does not have increased energy saving. All these phenomenon show that approximate computing is highly application and input dependent.

### C. Compare to Baseline

We compare **WOAxC** with the baseline method by performing the following steps: we use **WOAxC** to identify the approximation configuration under one dataset (e.g., butterfly), and then use this setting for another dataset (e.g., dogs), and then examine the output quality. The results are shown in Table III-V, which clearly shows that the baseline cannot guarantee the output quality.

Table III presents the case that the approximation configuration is identified based on butterfly dataset and then used on cats and dogs datasets. We observe that the approximation configuration can lead to four violations of dog datasets. Moreover, Table IV presents the case that the approximation configuration is identified based on cat dataset and is then used on butterfly and dog datasets. We can observe that not a single quality constraint being satisfied for butterfly and dog datasets. The similar situations can be observed in Table V where the approximation configuration identified under dog dataset leads to multiple violations of other datasets. In summary, a proper approximation configuration must be highly workload-dependent, otherwise, the output quality cannot be guaranteed.

### D. Discussion

The main contribution of this paper is the formulation of a constrained optimization problem that considers input workload and approximation configuration at the program level, and solve it using a two-phase approach. While the selection and tuning of heuristic algorithm is important to achieve good solution, it is not the main focus of this paper. We leave this direction open to follow up research, e.g., applying more advanced heuristic algorithms. Actually, without loss of generality, we changed the GA parameters from two-point operations (crossover and mutation) to three-point and four-point operations, and observe that **WOAxC**'s performance is similar. Our future work aims to extend **WOAxC** for approximate high-level synthesis by integrating **WOAxC** with more hardware-level constraints such as latency and area.

## VI. CONCLUSION

In this paper, we propose **WOAxC**, a workload-aware approach that can identify a proper approximation configuration

TABLE III
QUALITY VIOLATION UNDER BUTTERFLY-BASED APPROXIMATION SETTING

| | Sobel | | | Prewitt | | | Laplacian | | |
|---|---|---|---|---|---|---|---|---|---|
| **Quality constraint (%)** | butterfly | cat | dog | butterfly | cat | dog | butterfly | cat | dog |
| 5.0 | ✔ | ✔ | ✘(7.5) | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| 2.5 | ✔ | ✔ | ✘(7.5) | ✔ | ✔ | ✘(5.0) | ✔ | ✔ | ✔ |
| 0.0 | ✔ | ✔ | ✔ | ✔ | ✔ | ✘(5.0) | ✔ | ✔ | ✔ |

TABLE IV
QUALITY VIOLATION UNDER CAT-BASED APPROXIMATION SETTING

| | Sobel | | | Prewitt | | | Laplacian | | |
|---|---|---|---|---|---|---|---|---|---|
| **Quality constraint (%)** | butterfly | cat | dog | butterfly | cat | dog | butterfly | cat | dog |
| 5.0 | ✘(40.0) | ✔ | ✘(20.0) | ✘(47.5) | ✔ | ✘(17.5) | ✘(12.5) | ✔ | ✘(17.5) |
| 2.5 | ✘(32.5) | ✔ | ✘(20.0) | ✘(55.0) | ✔ | ✘(20.0) | ✘(12.5) | ✔ | ✘(17.5) |
| 0.0 | ✘(20.0) | ✔ | ✘(20.0) | ✘(52.5) | ✔ | ✘(22.5) | ✘(12.5) | ✔ | ✘(5.0) |

TABLE V
QUALITY VIOLATION UNDER DOG-BASED APPROXIMATION SETTING

| | Sobel | | | Prewitt | | | Laplacian | | |
|---|---|---|---|---|---|---|---|---|---|
| **Quality constraint (%)** | butterfly | cat | dog | butterfly | cat | dog | butterfly | cat | dog |
| 5.0 | ✔ | ✔ | ✔ | ✘(17.5) | ✔ | ✔ | ✘(12.5) | ✔ | ✔ |
| 2.5 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✘(12.5) | ✔ | ✔ |
| 0.0 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✘(10.0) | ✔ | ✔ |

to minimize energy consumption while meeting quality constraints. **WOAxC** is based on genetic algorithm to search the proper approximation configuration for a given program, input workload, and quality constraint. **WOAxC** has two key phases: pruning phase to reduce the search space, and searching phase to search the solution. Experimental results on three applications and three datasets show that **WOAxC** can enable on average, 79.6%, 77.4%, and 70.94% for quality loss of 5%, 2.5% and 0%, respectively. Comparisons with workload-unaware approaches highlight the importance of considering input workload in configuring approximate computing.

## VII. ACKNOWLEDGMENTS

This work was partially supported by NSF grant #2028889, and Villanova University Summer Grant. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## REFERENCES

[1] Dogs vs. cats. https://www.kaggle.com/c/dogs-vs-cats/.
[2] Omid Akbari, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. Rap-cla: A reconfigurable approximate carry look-ahead adder. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(8), 2016.
[3] Vahideh Akhlaghi et al. Lemax: learning-based energy consumption minimization in approximate computing with quality guarantee. In *DAC*, 2018.
[4] Vaibhav Gupta et al. Impact: imprecise adders for low-power approximate computing. In *ISLPED*, 2011.
[5] Soheil Hashemi et al. Drum: A dynamic range unbiased multiplier for approximate applications. In *ICCAD*, 2015.
[6] Xun Jiao et al. Clim: A cross-level workload-aware timing error prediction model for functional units. *IEEE Transactions on Computers*, 2017.
[7] Xun Jiao et al. Combining structural and timing errors in overclocked inexact speculative adders. In *DATE*, 2017.
[8] Xun Jiao et al. Energy-efficient neural networks using approximate computation reuse. In *DATE*, 2018.
[9] Chaofan Li, Wei Luo, Sachin S Sapatnekar, and Jiang Hu. Joint precision optimization and high level synthesis for approximate computing. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.
[10] Cong Liu et al. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *DATE*, 2014.
[11] Cong Liu et al. An analytical framework for evaluating the error characteristics of approximate adders. *IEEE Transactions on Computers*, 2015.
[12] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)*, 48(4):62, 2016.
[13] Vojtech Mrazek et al. autoax: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components. In *DAC*, 2019.
[14] Bharath Srinivas Prabakaran, Semeen Rehman, and Muhammad Shafique. Xbiosip: A methodology for approximate bio-signal processing at the edge. 2019.
[15] Shayan Tabatabaei-Nikkhah et al. Achilles: Accuracy-aware high-level synthesis considering online quality management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.
[16] Shaghayegh Vahdat, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. Letam: A low energy truncation-based approximate multiplier. *Computers & Electrical Engineering*, 63:1–17, 2017.
[17] Josiah Wang, Katja Markert, and Mark Everingham. Learning models for object recognition from natural language descriptions. In *Proceedings of the British Machine Vision Conference*, 2009.
[18] Chengwen Xu et al. On quality trade-off control for approximate computing using iterative training. In *Design Automation Conference (DAC)*, 2017.