LEVAX: An Input-Aware Learning-Based Error Model of Voltage-Scaled Functional Units

Xun Jiao, Dongning Ma, Wanli Chang, Yu Jiang

***Abstract*—As Moore's Law comes to an end and transistor scaling increasingly falls short in improving energy efficiency, alternative computing paradigms are direly needed. This need is further highlighted by the overwhelming increase in computing demand posed by emerging applications such as multimedia and data analysis. Fortunately, such driving workloads also present new opportunities since, thanks to their inherent error tolerance, they do not require completely accurate computations. Thus, by trading off accuracy for better performance or improved efficiency, approximate computing promises tremendous growth for future computing. Various approximation methods demonstrate the effectiveness of voltage scaling in functional units (FUs) for exploring this energy-error trade-off. Yet, while an accurate error model is critical for assessing the error behavior of voltage-scaled FUs and its effects on application quality, existing error models of voltage-scaled FUs overlook the effects of input data and error rate disparity among different bits. To tackle this challenge, we propose LEVAX, an input-aware learning-based error model of voltage-scaled FUs that can predict the timing error rate (TER) for each output bit. This model is trained using random forest methods, with input features and output labels extracted from gate-level simulations. To validate its effectiveness and demonstrate its prediction accuracy, we use LEVAX on various FUs. Across all bit positions, voltage levels, and FUs, LEVAX achieves, on average, a relative error of 1.20%. LEVAX also achieves an average per-voltage Root Mean Square Error (RMSE) of 1.03% and per-bit RMSE of 1.17%. Exposing this error rate even up to the application level, LEVAX can estimate the quality of four image processing applications under voltage scaling with an average accuracy of 97.9%. To the best of our knowledge, LEVAX is the first voltage scaling error model of FUs that can incorporate the effects of input data.**

I. Introduction

Many applications exhibit tolerance to computations with relaxed precision, e.g., multimedia processing [22], financial analysis [1], and machine learning applications [25]. Such intrinsic error tolerance is exploited by approximate computing to achieve improved performance and efficiency. Recently, various approximation methods have demonstrated the effectiveness of voltage scaling in functional units (FUs) for improved energy efficiency. This is mainly in adders and multipliers [7], [12], [19], [20] as these two operators represent the majority of computing workloads for many applications. For example, more than 98% of computations in convolutional neural networks (CNNs) are additions and multiplications, making them popular targets for voltage scaling seeking energy savings [30], [4], [10]. While effective, voltage scaling has the disadvantage of changing circuit delay, which causes timing errors that can lead to degradation of application quality. Thus, to enable precise error control, an accurate error model is of critical importance for predicting the error behavior of FUs and its effects on application quality.

Static timing analysis (STA) is a well-established method for measuring the timing performance of circuits under different operating conditions. However, STA cannot measure the timing errors under different input data because STA does not consider dynamic runtime information. For this reason, dynamic timing analysis, e.g., gate-level simulation, is adopted to measure the "true" timing performance under different input workloads [18], [16], [27], [23]. However, gate-level simulation has several shortcomings. First, accessing commercial simulation tools requires expensive licenses. Second, gate-level simulation is often prohibitively slow, which creates a bottleneck for the error-tolerant research [27]. Lastly, its setup requires multiple steps and domain knowledge in the electronic design automation (EDA) including logic synthesis, place-and-route, STA, and back-annotated simulation. This lengthy and tedious process can prohibit software developers and designers from taking this path [7], [12], [19], [20].

As a result, approximate computing researchers and designers have developed various error models including single bit flip [20], random values [19], and bit flip with uniform probability [7], [12]. Nonetheless, these error models are unable to fully capture the error behavior of circuits. First, they overlook the probable error rate disparity among different bit positions. For example, bit flip with uniform probability [7], [12] assumes that each bit has the same error rate. However, each bit locates on different circuit paths that probably have different delays. Second, these error models overlook the effects of input data (operands) in errors. Actually, input data have direct effects on timing errors because they can affect the dynamic path sensitization [11]. Thus, it is important to consider input data for predicting voltage scaling-induced errors. When used in approximate computing, the misinterpretation of error behavior can lead to either an aggressive voltage scaling strategy that may render unacceptable output quality or a conservative strategy that may not fully explore the approximation benefits.

To overcome the aforementioned limitations, we propose LEVAX, an input-aware learning-based error model for voltage-scaled FUs that can predict the TER of each bit position. First, under voltage scaling, we characterize the timing error behavior of each bit position of various FUs.

X. Jiao and D. Ma are with the Department of Electrical and Computer Engineering, Villanova University, Villanova, PA 19085, USA (E-mail: dma2@villanova.edu, xun.jiao@villanova.edu)

W. Chang is with the Department of Computer Science, University of York, UK (E-mail: wanli.chang@york.ac.uk).

Y. Jiang is with the School of Software, Tsinghua University, China. (E-mail: jy1989@mail.tsinghua.edu.cn).

Based on such characterization, we extract the useful features. We then apply random forest learning methods to fit the extracted features and the corresponding timing errors. This leads to LEVAX. Note that while this process involves EDA design flow and gate-level simulation, this is a one-time effort that can be done offline. Next, we use LEVAX to predict the TER of each bit position and compare it with simulation-based ground truth so as to evaluate the prediction accuracy. We then inject the LEVAX-predicted errors into several error-tolerant applications and estimate the corresponding application quality. The advantages of LEVAX are three-fold: (1) improved accuracy from considering input data and bit position; (2) accelerated execution (5X-45X) over gate-level simulation; (3) friendly interface for software developers and designers as LEVAX is a standalone Python module to be open-sourced for the research community.

Main Contributions:

- We characterize the timing error behavior of various FUs under a wide range of supply voltages. We analyze the effects of voltage scaling and input data on the timing errors, based on which we extract useful features to train an error model.
- We propose LEVAX, an input-aware learning-based error model for voltage-scaled FUs to predict the TER of each bit position. We evaluate several machine learning methods and, due to the accuracy and efficiency of lEVAX, choose the random forest method for its construction.
- We apply LEVAX to various types of FUs including logically exact FUs (INT_ADD, INT_MUL, FP_ADD, FP_MUL) and logically inexact FUs (XINT_ADD, XINT_MUL) under 28 different voltage levels. On average across these FUs and voltage levels, LEVAX can predict timing error rates with an RMSE (Root Mean Square Error) and relative error at around 1% compared to the gate-level simulation.
- We use LEVAX to estimate the quality of error-tolerant applications as either *acceptable* or *unacceptable* under a given voltage scaling strategy. The estimation accuracy is 97.9% on average. The high accuracy and fast execution make LEVAX a promising alternative for evaluating voltage scaling-induced TERs to commercial simulators.

Organization of the Paper: Section II describes the necessary background. Section II-C to Section V present the proposed model. The experimental results are reported in Section VI. Section VII discusses possible limitations and potential improvements. Section VIII summarizes the related work, and Section IX concludes the paper.

II. Background and Model Overview

In this section, we describe two root causes of timing errors: dynamic path sensitization and voltage scaling. We also present an overview of LEVAX.

A. Dynamic Path Sensitization

Timing errors occur when the circuit delay is beyond the clock period. As a matter of fact, the circuit delay is dynamic as shown in Fig. 1: under different input operands, the circuit delay will change. Under the initial state (case (a)), both of the inputs are 0. If the first input is changed from 0 to 1 (case (b)), then the circuit path denoted by the purple line will be sensitized, exhibiting a delay of 2.0ns. Later, if the second input is also changed from 0 to 1 (case (c)), then the circuit delay will become 1.5ns because the purple line-denoted circuit path is sensitized. Depending on the clock period, there could be different outcomes. If the clock period is set as 1.7ns, then case (b) will incur a timing error because 2.0ns is greater than 1.7ns. In contrast, case (c) will not have a timing error because 1.5ns is less than 1.7ns. To prevent such timing errors from happening, circuit designers typically set the clock period based on the critical path and via static timing analysis, which is greater than 2.0ns.

B. Voltage Scaling

As stated above, a conservative clock period is used for error-free operation. However, recent work has studied voltage scaling as a potential method of approximate computing and have demonstrated its effectiveness [7], [12], [19], [20]. Reducing the voltage improves the energy efficiency. Voltage scaling can, however, change the path delay in circuits [9]. Specifically, reducing the voltage can increase the path delay. For example, the delay of case (b) can be increased beyond 2.0ns, leading to timing errors even if the clock period is set to 2.0ns. This indicates that voltage scaling can lead to timing errors even if the clock period is set based on critical paths and via static timing analysis.

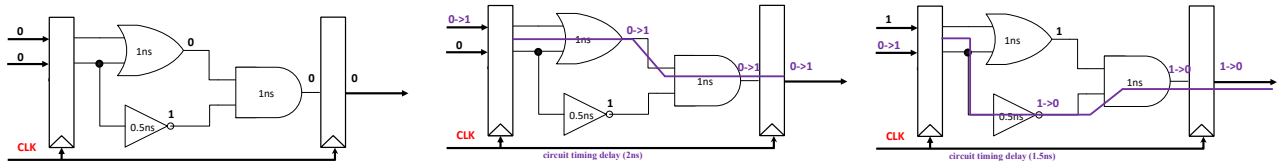
C. Proposed Model

LEVAX has three phases as illustrated in Fig. 2: *Data Collection*, *Model Construction*, and *Model Utilization*. The *Data Collection* phase uses gate-level simulations of post-layout FUs to collect the training data. The *Model Construction* phase uses the random forest method to train the LEVAX model, with features/labels extracted from training data. The *Model Utilization* phase predicts the TERs of FUs, which are then injected into applications to estimate the quality of applications. We compare the prediction results with simulation results to evaluate the accuracy of LEVAX. More details about the three phases are presented as follows.

III. Data Collection

A. Training Data Generation

It is impossible to generate an exhaustive training dataset for a regular-size circuit. For example, for a circuit with two 32-bit inputs, we need to generate 2^{64} input patterns to cover all possible scenario of a circuit. Thus, we propose to use machine learning methods, which are designed for learning the behaviors from limited input data. To evenly distribute the training data, we randomly generate $1M$ input data with each bit randomly assigned 0 or 1.



(a) Initial state (b) First input changes (c) Second input changes

Fig. 1: Different delay under different input

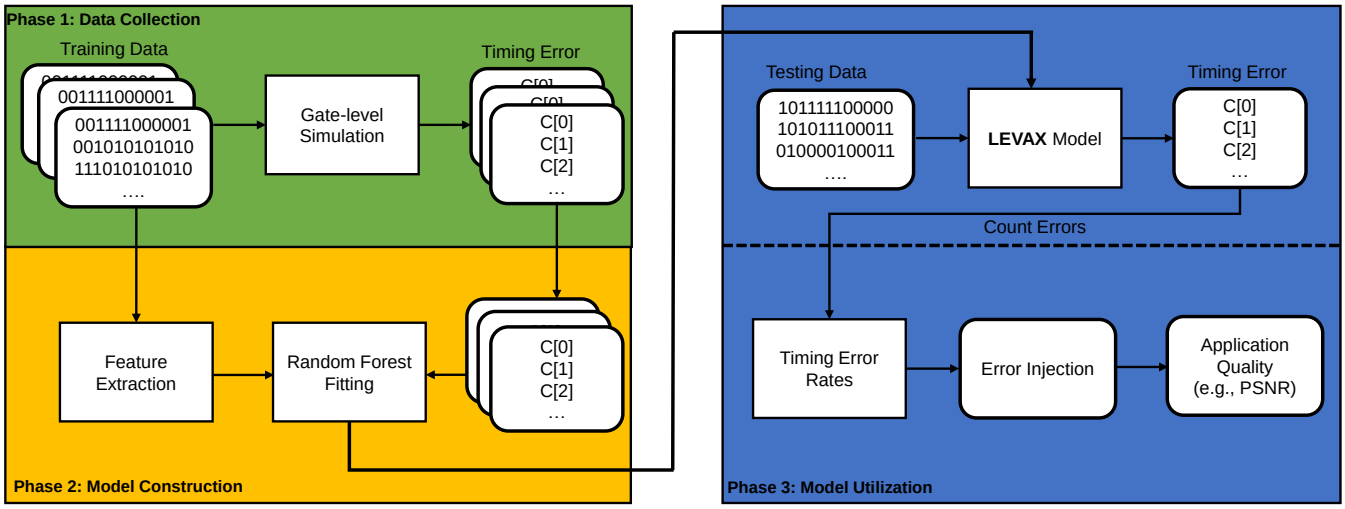


Fig. 2: LEVAX overview with three phases: 1) phase 1 generates timing errors under different voltages and input operands; b) phase 2 extracts useful features and applies random forest learning methods to train an error model; c) phase 3 estimates the application quality using the predicted timing error rates.

B. Timing Error Extraction

We use 32-bit integer and single-precision floating point units (FPUs) as our benchmarks, both generated from FloPoCo [6]: integer adder (INT_ADD), integer multiplier (INT_MUL), floating point adder (FP_ADD), and floating point multiplier (FP_MUL), all implemented in synthesizable VHDL. FPUs follow the IEEE-754 standard and provides more complex structures compared to their integer counterparts. To better evaluate the robustness of LEVAX, we change the data types and circuit topology. We also use approximate FUs from lpACLib [21], an open-source library for low-power approximate integer adders (XINT_ADD) and multipliers (XINT_MUL). We extract timing errors through a standard gate-level simulation process, which is divided into several steps.

We use *Synopsys Design Compiler* to synthesize the VHDL codes and *Synopsys IC Compiler* to place and route the design, in TSMC 45nm technology. Then, we use *Synopsys PrimeTime* to conduct static timing analysis, and inject different voltages into the design. This results in the corresponding standard delay format (SDF) files that contain the delay information under each voltage. We use a wide range of voltage from $0.72V$ to $0.99V$ with a step size of $0.01V$. Thus, we use 28 voltage levels in total. *Mentor Graphics ModelSim* is used to perform SDF back-annotation gate-level simulation. For each voltage, we perform gate-level simulation to characterize the corresponding TERs of each bit. The input stimuli for simulation is 1 million random data. (This random data set is also served as our training data set).

At each clock cycle, we identify the timing error at each bit by comparing each output bit with the clean output. Specifically, we identify a timing error if the simulation output is not matched with the clean output. Thus at cycle t, we classify the output bit $C[t]$ to two classes: C_c means timing correct and C_e means timing erroneous.

IV. Model Construction

A. Feature Extraction

Timing errors are generated when the circuit delay is beyond the circuit clock period. As shown in Section II, circuit delay can be affected by two classes of factors. The first class is the circuit-associated parameters such as voltage scaling. In this paper, we focus on using voltage scaling as an approximation strategy. Thus, we assume the other parameters (process, temperature, aging, etc.) remain unchanged. The second class of factors that can affect the circuit delay is the input data. In fact, while input data do not directly affect the delay of cells or paths, they can determine which circuit paths are sensitized. Consequently, the delay of the longest sensitized path becomes the "actual" circuit delay. Therefore, the joint effects of voltage and input data determine the circuit delay and hence the timing errors.

Thus, to predict the timing errors, we need to consider both voltage and input data. Since the voltage scaling is

typically performed at limited discrete levels within a specific range [29], it is possible to train the model using all possible voltage levels. In this paper, we consider the range of supply voltages from 0.72V to 1.00V with a step size of 0.01V.

However, it is impossible to be exhaustive and use all potential input data because two 32-bit input data can make 2^{64} different possible vectors. Therefore, the critical accomplishment of LEVAX is to predict timing errors under unseen input data. That is, the model needs to capture the "actual" circuit delay by learning the path sensitization behavior under unseen input data. Therefore, we also use random data as our training dataset because we want to maximally randomize the dynamic path sensitization behavior so we can capture the sensitization behavior under unseen data.

To do this, we start with a path sensitization analysis. According to [3], a sensitized path would have all of its nodes toggled. For a node to be toggled, the current signal value at the node needs to be different than the previous one. Thus, for a combinational circuit, both the current input and the previous input determine whether a node gets toggled and hence the path sensitization. That is, the joint effects of previous and current input determine the sensitized paths and the "actual" circuit delay. To verify this hypothesis, we conduct two experiments.

- Experiment 1: fix the input at cycle t ($x[t]$) but randomly vary the input at cycle $t-1$ ($x[t-1]$), where t is randomly selected throughout the simulation trace;
- Experiment 2: fix both the input of cycle t ($x[t]$) as well as $t-1$ ($x[t-1]$), where t is randomly selected throughout the simulation trace;

We observe that in Experiment 1, the timing error behavior of the bits in output $y[t]$ varies irregularly, e.g., bit position 5 in $y[10]$ is C_e while bit position 5 in $y[30]$ is C_c. While in Experiment 2, all the bits in output $y[t]$ are fixed in timing error behavior, e.g., the same bit positions in $y[10]$, $y[30]$, ... $y[t]$ are all either C_e or C_c. This verifies the previous hypothesis of joint effects on previous and current input. Through such analysis and experiments, we set the input features as $\{x[t-1], x[t], V\}$ and set $C[t]$ at each bit position as our label.

B. Random Forest-based Training

The training data is fed into gate-level simulation to generate timing error labels. The training data is used in a *feature extraction* module to extract useful features. We then use random forest methods to fit these data and train LEVAX. (we compare the random forest method with other machine learning methods in Section VI-B).

Random forest is an ensemble-learning method that is composed of multiple decision trees. Decision tree methods use Boolean logic to construct a set of learning decision rules from training data. The biggest advantage of decision trees is their easy interpretation. However, decision trees can be overfitted easily by learning irregular patterns with a large variance. Compared to decision tree, the random forest method controls the overfitting by constructing multiple decision trees and taking a weighted vote of their predictions. In our case, the random forest method constructs trees (decision rules) considering different bit positions and the interaction between them.

$$S = \begin{bmatrix} f_{1A} & f_{1B} & f_{1C} & ... & f_{1N} & C[1] \\ f_{2A} & f_{2B} & f_{2C} & ... & f_{2N} & C[2] \\ \vdots & \vdots & \vdots & ... & \vdots & \vdots \\ f_{dA} & f_{dB} & f_{dC} & ... & f_{dN} & C[d] \end{bmatrix} \quad (1)$$

Our training features and labels are illustrated in 1. The feature vectors are $\{f_{1A}, f_{1B}, ...f_{1N}\}$, $\{f_{2A}, f_{2B}, ...f_{2N}\}$, etc., where N is the number of bits in the combined input pattern, e.g., 128 for a 32-bit FU. Each feature, e.g., f_{1A}, is a binary value of a bit. Therefore, for an FU with two 32-bit inputs, the feature size is $4 * 32 = 128$ dimensions (including preceding and current inputs) and an additional dimension for voltage. The labels are $C[1]$, $C[2]$, etc., where each label, e.g., $C[1]$, is either *correct* or *erroneous*. If we use M decision trees, then random forest will split this whole S matrix into an M sub-matrix and fit a decision tree for each sub-matrix. The machine learning library is adopted from *Scikit-Learn* [15]. We use the default parameter configurations of random forest methods in the Scikit-Learn module, e.g., 10 trees in the forest.

V. Model Utilization

LEVAX's goal is to predict the timing error rate for each bit, which can then be used as the bit flip probability for injecting errors into applications [7], [12], [19], [20].

A. Timing Error Prediction

Since LEVAX's actual predicted result is the timing class of each bit at each clock cycle, i.e., $C[t]$, we just need to count the portion of timing errors to calculate the timing error rates of bit i under voltage v as follows:

$$TER(i,v) = \frac{\#C_e(i,v)}{\#C_e(i,v) + \#C_c(i,v)} \quad (2)$$

where $\#C_e$ and $\#C_c$ are the number of erroneous classes and the number of correct classes respectively. We also use gate-level simulation (gls) to calculate the timing error rates of each bit under each voltage. We compute the relative error for each bit under each voltage level as follows:

$$RE(i,v) = \frac{|TER_{levax}(i,v) - TER_{gls}(i,v)|}{TER_{gls}(i,v)} \quad (3)$$

where $TER_{levax}(i,v)$ and $TER_{gls}(i,v)$ are LEVAX predicted TER and simulation-based TER of bit i at voltage v, respectively.

We also use root mean squared error (RMSE) to evaluate the modeling accuracy of LEVAX. We compute per-bit RMSE across all voltages and per-voltage RMSE across all bits:

$$RMSE(i) = \sqrt{\frac{1}{28}\sum_{v=1}^{28}\Big(TER_{levax}(i,v) - TER_{gls}(i,v)\Big)^2}$$
$$RMSE(v) = \sqrt{\frac{1}{32}\sum_{i=1}^{32}\Big(TER_{levax}(i,v) - TER_{gls}(i,v)\Big)^2} \quad (4)$$

B. Application Quality Estimation

LEVAX predicts the timing error rates for each bit under a given voltage. Then, we inject these timing errors into the corresponding arithmetic operations in applications and evaluate the resulting application quality. We also inject the simulation-based errors to the application and obtain the resulting application quality. For image processing applications, the quality is typically considered *acceptable* if $PSNR \geq 30dB$, and otherwise *unacceptable* [2]. Thus, we compare the estimated application quality under LEVAX-based errors and simulation-based errors to evaluate the accuracy of LEVAX in estimating application quality.

VI. Experimental Results

A. Timing Error Characterization

Fig. 3 presents the bit-level TERs of all six FUs, while for the sake of readability we draw only five evenly-distributed voltage levels from which we can observe several important facts. First, the TER varies dramatically with different bit positions. For example, in INT_MUL under 0.72V, bit 0 exhibits almost zero TER while bit position 20 has 60% TER. This bit-level TER disparity justifies the importance of predicting the TER for each bit position. Second, the TER varies dramatically with different voltage levels. For example, the TER of bit 0 of INT_ADD under 0.72V is around 50% while under 0.90V is around 0%. This significant disparity in TERs among different voltage levels justifies the necessity of developing such a voltage scaling model. Third, the TER varies dramatically with different FUs. For example, INT_ADD has TERs less than 10% for all bits while FP_MUL has TERs greater than 10% for all bits.

B. Accuracy of Timing Error Prediction

The testing data is 1 million unseen random data. Fig. 4 presents the reliability (1-TER) based on LEVAX and the actual TERs derived by gate-level simulations (gls), but due to space limitations, under only four evenly distributed voltages — 0.76V, 0.82V, 0.88V and 0.94V. Nonetheless, these four demonstrate that the LEVAX-based TER closely follows the trend of gls-based TERs. The figure also presents the relative error for each bit (bar denoted). We can see that almost all of them are below 5%. Actually, the average relative error across all FUs, voltages, and bit positions is 1.20%.

We include a holistic evaluation and calculate the per-voltage RMSE and per-bit RMSE using equation 4 as illustrated in Table I and Table II. We can observe that LEVAX's prediction deviates only slightly from the gls-based TERs. The per-voltage RMSE as shown in Table I ranges from 0 to 2.793%, with most values less than 1%. The average per-voltage RMSE across all functional units is 1.03%. Table II presents the per-bit RMSE, which ranges from 0.009% to 2.213%. The average per-bit RMSE across all functional units is 1.17%.

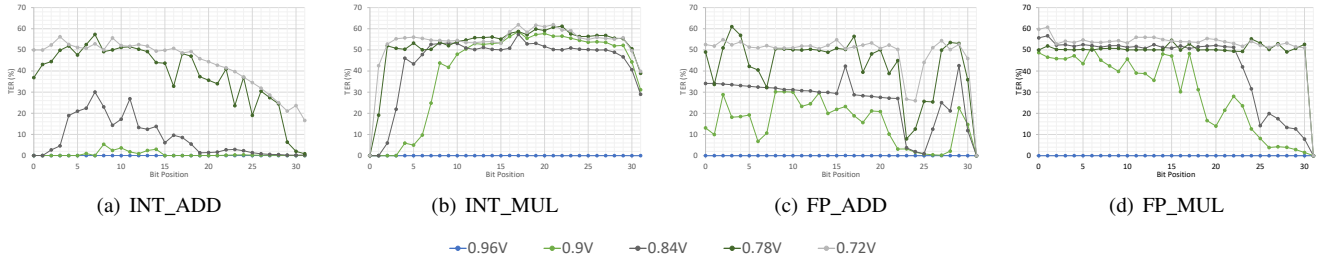
We also measure the training and inference time of LEVAX and compare it with the gate-level simulation. To simulate 1 million data on a machine with 2-core Intel(R) Xeon(R) CPU E5504@2.00GHz and 50GB memory, *Mentor Graphics Modelsim* typically takes more than 100s for integer units, 580s for FP_ADD, and 935s for FP_MUL. LEVAX takes less than 40s to train integer or floating point units and takes around 20s to perform inference for 1 million data. Considering that the training is a one-time process that can be done offline, LEVAX can characterize the timing errors 5X-45X faster than gate-level simulation. Actually, after only one session of offline training, we dump the LEVAX as a standalone Python module using the *joblib* module. The high prediction accuracy and significant acceleration over gate-level simulation make it possible to use LEVAX as an alternative to commercial simulators.

We also evaluate some widely-used machine learning methods such as logistic regression, support vector machine, and nearest neighbor (all with default parameter settings). Nearest neighbor provides useful theoretical properties [5] and has limited parameters to train. Logistic regression and support vector machine can learn and assign different weights to each feature. Compared with these methods, random forest can achieve significantly higher prediction accuracy with a 10X smaller RMSE. Actually, random forest fits our task better than other methods because random forest can better represent how paths are sensitized by different bits of input operands. First, different bits play different roles in sensitizing paths. Compared with nearest neighbor which does not distinguish between bits, random forest can interpret the significance disparity between different bits. Second, the role of each bit will be affected by other bits. Compared with support vector machine and logistic regression which assigns fixed weight to each bit, random forest considers the interactions between different bits.

C. Accuracy of Application Quality Estimation

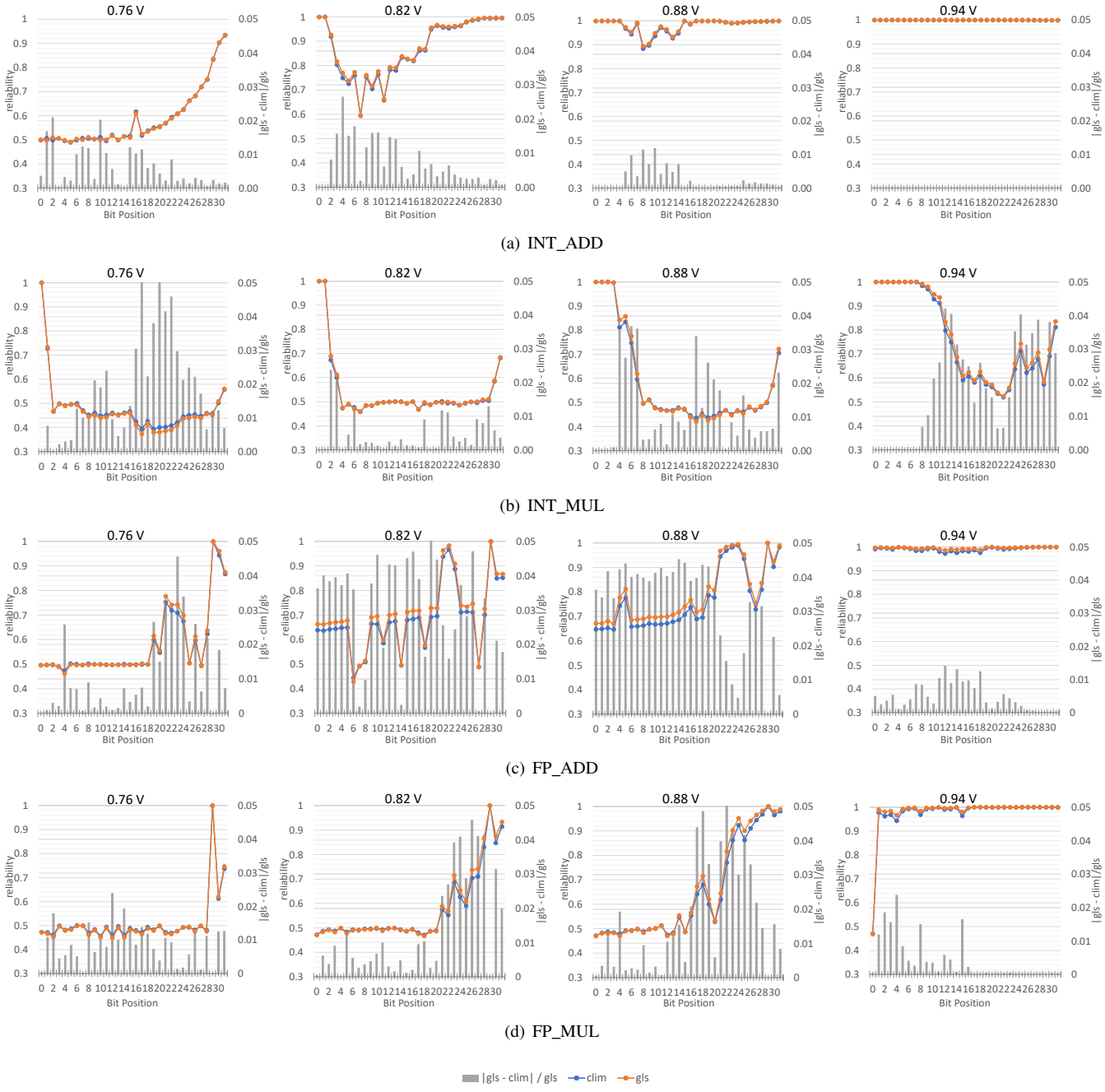
We use LEVAX-based error models to evaluate the effect of voltage scaling on application quality via error injection. We consider four widely-used error-tolerant image processing applications, Sobel filter, Roberts filter, Sharpen filter, and Scharr filter. We profile the application specific input data of FUs using an architectural simulator *Multi2Sim* [24]. These input data are fed into gate-level simulation (gls) and LEVAX to measure (predict) their TERs under all 28 voltage scaling levels. Then, we inject the corresponding errors to *Multi2Sim* [24] for quality estimation by modifying the source code of *Multi2Sim*. Because the timing errors manifest as bit flips, we then locate each target instruction and flip each output bit with a probability based on the bit-level TER. For example, if the TER for the 5^{th} bit of FP_MUL is 1%, we randomly flip such bit with a probability of 1%. This is similar to existing studies in approximate computing [7], [12], [19], [20]. We use butterfly image dataset from Caltech-101 [8] as the input data to *Multi2Sim*.

We use four different sources for voltage scaling-induced errors: gls (gate-level simulation) as ground truth, LEVAX, single-bit flip [20], and uniform bit flip [7]. We consider all the voltages from 0.72V to 0.99V with a step size of 0.01V. For each tested image under each voltage, we generate



(a) INT_ADD (b) INT_MUL (c) FP_ADD (d) FP_MUL

Fig. 3: Timing error rates of different bit positions



(a) INT_ADD

(b) INT_MUL

(c) FP_ADD

(d) FP_MUL

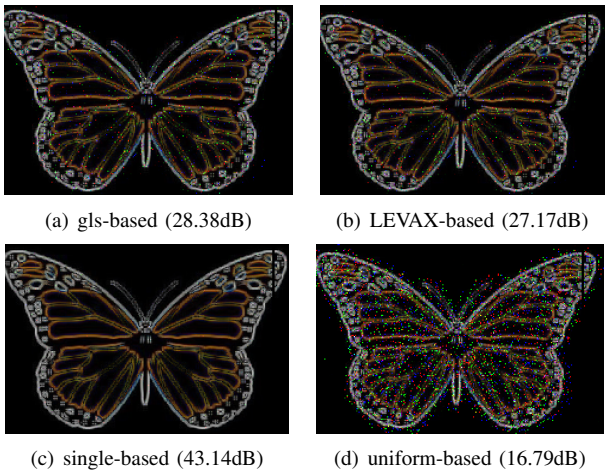
Fig. 4: Performance of LEVAX prediction

TABLE I: Per-Voltage RMSE of four functional units.

	0.72V	0.73V	0.74V	0.75V	0.76V	0.77V	0.78V	0.79V	0.80V	0.81V	0.82V	0.83V	0.84V	0.85V
INT ADD	0.604%	0.569%	0.388%	0.340%	0.442%	0.392%	0.474%	0.490%	0.645%	0.619%	0.773%	0.638%	0.707%	0.812%
INT MUL	0.910%	1.131%	1.047%	0.983%	0.962%	1.023%	0.817%	0.768%	0.495%	0.579%	0.457%	0.672%	0.614%	0.664%
FP ADD	0.964%	1.138%	1.163%	1.198%	1.153%	1.448%	1.578%	1.647%	1.977%	2.220%	2.400%	2.487%	2.625%	2.624%
FP MUL	0.748%	0.688%	0.750%	0.719%	0.520%	0.479%	0.429%	0.551%	0.731%	1.063%	1.468%	1.768%	1.866%	1.768%
	0.86V	0.87V	0.88V	0.89V	0.90V	0.91V	0.92V	0.93V	0.94V	0.95V	0.96V	0.97V	0.98V	0.99V
INT ADD	0.582%	0.570%	0.418%	0.370%	0.230%	0.116%	0.049%	0.013%	0.004%	0.002%	0.001%	0.001%	0.000%	0.000%
INT MUL	0.818%	0.951%	1.089%	1.109%	1.036%	1.141%	1.260%	1.541%	1.729%	2.441%	2.267%	1.512%	0.577%	0.104%
FP ADD	2.627%	2.680%	2.611%	2.729%	2.740%	2.625%	2.107%	1.283%	0.634%	0.194%	0.023%	0.001%	0.000%	0.000%
FP MUL	1.746%	1.751%	1.885%	2.035%	2.309%	2.793%	2.709%	1.867%	0.769%	0.216%	0.027%	0.000%	0.000%	0.000%

TABLE II: Per-bit RMSE of four functional units.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
INT ADD	0.181%	0.365%	0.591%	0.476%	0.625%	0.692%	0.725%	0.525%	0.673%	0.810%	0.756%	0.516%	0.498%	0.476%	0.497%	0.328%
INT MUL	0.006%	0.258%	0.528%	0.878%	1.131%	1.057%	1.029%	1.257%	1.087%	1.188%	1.019%	1.073%	1.154%	1.100%	0.958%	1.201%
FP ADD	1.696%	1.872%	1.819%	1.959%	2.009%	2.121%	1.874%	1.830%	1.851%	1.844%	2.044%	1.766%	2.059%	2.011%	1.695%	2.067%
FP MUL	0.883%	1.195%	1.278%	1.252%	1.208%	1.203%	1.105%	1.219%	1.198%	1.143%	1.147%	1.106%	1.142%	1.188%	1.221%	1.255%
	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
INT ADD	0.378%	0.445%	0.306%	0.369%	0.194%	0.275%	0.261%	0.197%	0.196%	0.211%	0.187%	0.171%	0.139%	0.130%	0.512%	0.417%
INT MUL	1.336%	1.369%	1.093%	1.253%	1.138%	1.340%	1.203%	1.249%	1.484%	1.296%	1.274%	1.229%	1.192%	1.160%	1.257%	1.413%
FP ADD	2.175%	2.035%	1.916%	2.254%	2.050%	2.237%	1.939%	1.914%	1.951%	1.588%	1.694%	1.486%	1.679%	0.000%	1.798%	1.104%
FP MUL	1.305%	1.473%	1.538%	1.402%	1.152%	1.451%	2.213%	2.296%	1.908%	2.087%	1.936%	1.737%	1.628%	0.000%	1.535%	1.067%



(a) gls-based (28.38dB) (b) LEVAX-based (27.17dB)

(c) single-based (43.14dB) (d) uniform-based (16.79dB)

Fig. 5: Output quality based on gls, LEVAX, single, and uniform models under 0.94V

four different versions of output image — gls-based, LEVAX-based, single-based, and uniform-based. If any model-based output image has the same quality classification as gls-based, we count that as a correct estimation.

Fig. 5 shows an example of an output image of Roberts filter under 0.94V. We can see that the LEVAX-based image quality is very similar to that of the gls-based, where the pixel noise distribution is also very similar between these two images. (Note that this is hard to read in printed papers). The single-based image has little noise while the uniform-based image is full of noisy pixels. More specially, both gls-based and LEVAX-based models lead to a PSNR around 28dB while the single-based model leads to a PSNR at 43dB and the uniform-based model leads to 17dB. LEVAX achieves the same estimation with gls because the timing error rates are very close. The single-bit model only assumes one bit flipped at each time thus significantly underestimating the errors. In contrast, the uniform model overestimates the errors because it assumes the same error rates for each bit. Generally speaking, the single-bit model tends to underestimate the errors while the uniform model tends to overestimate the errors.

In this example, because gls-based quality measurement is *unacceptable* ($25dB \leq 30dB$), the uniform model also makes a correct prediction. As shown in Table III, we use the same process for all the voltages and have seen that LEVAX-based quality estimation is on average 97.9% accurate, while single-based and uniform-based models only achieve 71.48% and 89.5% estimation accuracy. The uniform-based model achieves this degree of accuracy because it tends to overestimate the errors and predict an image quality to be *unacceptable*, while the actual image quality also drops rapidly as the voltage is down-scaled.

TABLE III: Application quality estimation using three models.

Application	LEVAX	single-bit	uniform
Sobel	96.6%	75.8%	78.3%
Robert	97.5%	57.5%	95.8%
Schaar	99.0%	88.5%	86.4%
Sharpen	98.5%	64.1%	97.5%

D. Comparison with Related Works

We compare LEVAX with some existing studies in modeling voltage-induced timing errors.

- Single-bit [20]: The single-bit model only assumes one bit is erroneous (and will be flipped) when there is a timing error, and, hence, may underestimate the impact of timing errors. As shown in Fig. 3, each bit has its own TER and, hence, multiple bits can have errors simultaneously. Therefore, LEVAX considers and predicts TER for each bit position.
- Uniform [7]: In contrast to the single-bit model, the uniform model assumes the same error rates for each bit. Similarly, as shown in Fig. 3, we can see that each bit can have different TERs. Therefore, LEVAX treats each bit position differently and predicts their TER independently.
- B-Hive [23]: LEVAX is significantly different than B-Hive in several major aspects: (1) B-Hive divides the output into five classes based on the relationship between previous output and current output, e.g., previous observed class, previous correct class, etc. It is not straightforward to measure whether an output has timing errors based on such classification. LEVAX, instead, forms a binary classification problem and intuitively predict each output as either $correct$ or $erroneous$, hence directly predicting timing error rates. (2) B-Hive is based on the strong assumption that each output is visible to the model and can be used to predict next output. This assumption is not practical because it requires running (or simulating) the circuit and perform a value check for every clock cycle. LEVAX, instead, only uses input information and does not need to simulate or check the output value. (3) B-Hive uses a frequency-of-occurrence-based method to model the errors and does not consider the effects of input operands. LEVAX, on the other hand, explores the root cause of timing errors, which is related to the dynamic path sensitization. As shown in Section II, path sensitization is determined by the input operands, based on which we predict errors. We use B-Hive's frequency-of-occurrence-based method to predict timing errors of all four FUs and compare with LEVAX, as illustrated in Table IV. We can observe that LEVAX's RMSE is significantly smaller than B-Hive.

TABLE IV: Comparison between LEVAX and B-Hive.

FU	Per-voltage RMSE		Per-bit RMSE	
	LEVAX	B-Hive	LEVAX	B-Hive
INT_ADD	0.366%	16.586%	0.410%	23.298%
INT_MUL	1.025%	35.421%	1.100%	37.301%
FP_ADD	1.603%	23.617%	1.823%	27.532%
FP_MUL	1.131%	30.710%	1.358%	35.245%

VII. Discussion

Scope: Voltage scaling, as an approximation method, is mainly performed on arithmetic units [7], [12], [19], [20], i.e., FUs, because many modern applications can tolerate inaccuracy in computation. However, approximate computing is rarely performed on control operations because that can lead to catastrophic results in application quality or even system corruption such as segmentation fault [28]. Thus, in this work, we specifically focus on the error modeling for FUs rather than other parts of the system such as memory. The same approach may not be generalized to them because they may not follow the same path sensitization principle as FUs.

Learning Methods: While there is a large parameter space and a wide range of learning methods we can use, the main contribution of this paper is the formulation of the error modeling by considering input data and bit positions. In this work, the default parameter configurations of Scikit-Learn [15] have already shown a high prediction accuracy. Thus, we leave the tuning of machine learning parameters for future work.

Future Work: Possible future work may entail mainly two areas of exploration: 1) developing a system-level error model for processors and chips; the challenges in developing such a model lie in the even more complicated dynamic path sensitization behavior and the control paths activation, for example, how to define proper features for multiple pipeline stages and 2) crafting a more selective training dataset with even more coverage on dynamic path sensitization; for example, we can consider automatic test pattern generation (ATPG) tools used in circuit testing to improve the process.

VIII. Related Work

Approximate computing improves the operational efficiency of computing systems by allowing occasional errors so long as the application output quality is acceptable. Voltage scaling on FUs has been shown to be effective in exploiting such efficiency-accuracy tradeoff. Voltage scaling-induced error models that are currently being used in approximate computing include single bit flip [20], random values [19], and bit flip with uniform probability [7], [12]. One randomly chosen bit is flipped if there is a voltage scaling-induced timing error [20]. Single bit flip, last value, and random value models are considered in [19]. The last value model (or random value model) means that the last output of a specific operation (or a random value) will be returned if the operation has timing errors that are caused by voltage scaling. The bit flip model is based on a per-component probability that is used in [7]. Another study models the error rate as a function of supply voltage only and does not differentiate the error rate between each bit position [12]. B-Hive [23] predicts bit-level error rates using a frequency-of-occurrence-based Modeling.

Another set of studies has used analytical and learning models to estimate voltage scaling-induced errors of approximate circuits. A simulation-based approach is used in [13] to evaluate several statistical error metrics such as error distance and the mean error distance for inexact adders. Han et.al proposed an analytical model to evaluate similar statistical error metrics for approximate adders [14]. These works mainly focus on evaluating the error caused by inexact logic design rather than those caused by voltage scaling, which can be more random and lack statistical properties. MACACO uses Monte-Carlo simulation to evaluate several metrics of voltage scaling-induced errors such as worst-case error, average-case error,

and error distribution [26]. Rahimi et.al proposed a variation-aware error model for functional units [17]. However, all of these works do not consider the error rate disparity among different bit positions and mainly focus on integer functional units. B-Hive [23] divides the bit errors into five classes based on the relationship between previous output and current output and uses a frequency-of-occurrence-based method to model the errors. It does not consider the effects of input operands and only uses the statistical metric to predict the voltage scaling-induced errors.

Our work is different from these aforementioned works in two ways: 1) LEVAX considers the effects of input data in predicting voltage scaling-induced errors. It requires no prior knowledge of input distribution and output history, and it can predict errors for any unseen input. 2) LEVAX considers the error behavior disparity among different bit positions and is able to predict the timing error rate for each bit position.

IX. Conclusion

In this paper, we propose LEVAX, an input-aware learning-based model that can predict timing error rates of each bit in functional units. We perform extensive error characterization under various voltage levels and extract useful features from the input data to predict the timing errors. We apply random forest methods to train LEVAX. LEVAX can obtain a high prediction accuracy, i.e., within a small deviation from gate-level simulation and with significant acceleration. We further use LEVAX to estimate the application quality for four image-processing applications and LEVAX outperforms the existing error models in estimation accuracy. The high accuracy and fast computing speed make LEVAX an alternative to commercial simulation tools for evaluating the voltage scaling-induced errors. Our future work focuses on developing error models for even more complicated circuits and processors.

References

[1] Woongki Baek and Trishul M Chilimbi. Green: a framework for supporting energy-conscious programming using controlled approximation. In *ACM Sigplan Notices*, volume 45, pages 198–209. ACM, 2010.

[2] Mauro Barni. *Document and Image compression*. CRC press, 2006.

[3] Michael Bushnell and Vishwani Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, volume 17. Springer Science & Business Media, 2004.

[4] Wonseok Choi, Dongyeob Shin, Jongsun Park, and Swaroop Ghosh. Sensitivity based error resilient techniques for energy efficient deep neural network accelerators. In *Proceedings of the 56th Annual Design Automation Conference 2019*, page 204. ACM, 2019.

[5] Thomas M Cover and Peter E Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[6] Florent De Dinechin and Bogdan Pasca. Designing custom arithmetic data paths with flopoco. *IEEE Design & Test of Computers*, 28(4):18–27, 2011.

[7] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Architecture support for disciplined approximate programming. In *ACM SIGPLAN Notices*, volume 47, pages 301–312. ACM, 2012.

[8] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer vision and Image understanding*, 106(1):59–70, 2007.

[9] Ricardo Gonzalez, Benjamin M Gordon, and Mark A Horowitz. Supply and threshold voltage scaling for low power cmos. *IEEE Journal of Solid-State Circuits*, 32(8):1210–1216, 1997.

[10] Xun Jiao, Mulong Luo, Jeng-Hau Lin, and Rajesh K Gupta. An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 945–950. IEEE Press, 2017.

[11] Veit B Kleeberger, Petra R Maier, and Ulf Schlichtmann. Workload-and instruction-aware timing analysis: The missing link between technology and system-level resilience. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[12] Evgeni Krimer, Patrick Chiang, and Mattan Erez. Lane decoupling for improving the timing-error resiliency of wide-simd architectures. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 237–248. IEEE Computer Society, 2012.

[13] Jinghang Liang, Jie Han, and Fabrizio Lombardi. New metrics for the reliability of approximate and probabilistic adders. *IEEE Transactions on computers*, 62(9):1760–1771, 2013.

[14] Cong Liu, Jie Han, and Fabrizio Lombardi. An analytical framework for evaluating the error characteristics of approximate adders. *IEEE Transactions on Computers*, 64(5):1268–1281, 2015.

[15] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[16] Abbas Rahimi, Luca Benini, and Rajesh K Gupta. Application-adaptive guardbanding to mitigate static and dynamic variability. *IEEE Transactions on Computers*, 63(9):2160–2173, 2013.

[17] Abbas Rahimi, Luca Benini, and Rajesh K Gupta. Hierarchically focused guardbanding: An adaptive approach to mitigate pvt variations and aging. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1695–1700. IEEE, 2013.

[18] Sanghamitra Roy and Koushik Chakraborty. Predicting timing violations through instruction-level path sensitization analysis. In *Proceedings of the 49th Annual Design Automation Conference*, pages 1074–1081. ACM, 2012.

[19] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Notices*, volume 46, pages 164–174. ACM, 2011.

[20] John Sartori, Joseph Sloan, and Rakesh Kumar. Stochastic computing: embracing errors in architectureand design of processors and applications. In *Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems*, pages 135–144. ACM, 2011.

[21] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jörg Henkel. A low latency generic accuracy configurable adder. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[22] Renée St Amant, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, Hadi Esmaeilzadeh, Arjang Hassibi, Luis Ceze, and Doug Burger. General-purpose code acceleration with limited-precision analog computation. *ACM SIGARCH Computer Architecture News*, 42(3):505–516, 2014.

[23] G Tziantzioulis, AM Gok, SM Faisal, Nikolaos Hardavellas, S Ogrenci-Memik, and Srinivasan Parthasarathy. b-hive: A bit-level history-based error model with value correlation for voltage-scaled integer and floating point units. In *Proceedings of the 52nd Annual Design Automation Conference*, page 105. ACM, 2015.

[24] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. Multi2sim: a simulation framework for cpu-gpu computing. In *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 335–344. IEEE, 2012.

[25] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. Axnn: energy-efficient neuromorphic systems using approximate computing. In *Proceedings of the 2014 international symposium on Low power electronics and design*, pages 27–32. ACM, 2014.

[26] Rangharajan Venkatesan, Amit Agarwal, Kaushik Roy, and Anand Raghunathan. Macaco: Modeling and analysis of circuits for approximate computing. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 667–673. IEEE, 2011.

[27] Jing Xin and Russ Joseph. Identifying and predicting timing-critical instructions to boost timing speculation. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 128–139. ACM, 2011.

[28] Yavuz Yetim, Margaret Martonosi, and Sharad Malik. Extracting useful computation from error-prone processors for streaming applications.

In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 202–207. EDA Consortium, 2013.

[29] Bo Zhai, David Blaauw, Dennis Sylvester, Dennis Sylvester, and Krisztian Flautner. Theoretical and practical limits of dynamic voltage scaling. In *Proceedings of the 41st annual Design Automation Conference*, pages 868–873. ACM, 2004.

[30] Jeff Zhang, Kartheek Rangineni, Zahra Ghodsi, and Siddharth Garg. Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators. In *Proceedings of the 55th Annual Design Automation Conference*, page 19. ACM, 2018.