TEVoT: Timing Error Modeling of Functional Units under Dynamic Voltage and Temperature Variations

Xun Jiao[‡], Dongning Ma[‡], Wanli Chang[§], Yu Jiang[*]
[‡]Villanova University, [§]University of York, [*]Tsinghua University
{xjiao, dma2}@villanova.edu, wanli.chang@york.ac.uk, jiangyu198964@126.com

Abstract*—With the continuous scaling of CMOS technology, microelectronic circuits are increasingly susceptible to microelectronic variations such as variations in operating conditions. Such variations can cause delay uncertainty in microelectronic circuits, leading to *timing errors*. Circuit designers typically combat these errors using conservative guardbands in the circuit and architectural design, which can, however, cause significant loss of operational efficiency. In this paper, we propose TEVoT, a supervised learning model that can predict the timing errors of functional units (FUs) under different operating conditions, clock speeds, and input workload. We perform dynamic timing analysis to characterize the delay variations of FUs under different conditions, based on which we collect training data. We then extract useful features from training data and apply supervised learning methods to establish TEVoT. Across 100 different operating conditions, 4 widely-used FUs, 3 clocking speeds, and 3 datasets, TEVoT achieves an average prediction accuracy at 98.25% and is 100X faster than gate-level simulation. We further use TEVoT to estimate application output quality under different operating conditions by exposing circuit-level timing errors to application level. TEVoT achieves an average estimation accuracy at 97% for two image processing applications across 100 operating conditions.

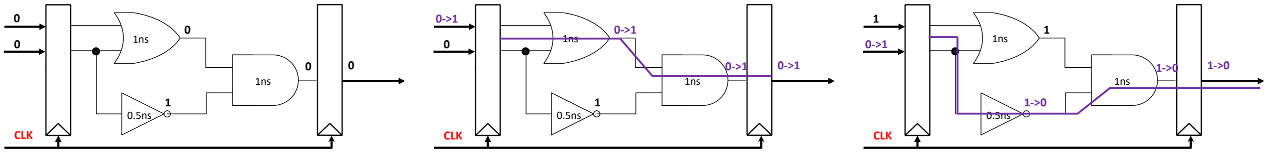
I. Introduction

As the transistor size scales down to deep nanometer era, microelectronic circuits are increasingly susceptible to microelectronic variability [10]. Generally speaking, microelectronic variability arises from various sources such as operating conditions, manufacturing, or aging [10]. Due to the burgeoning use of microelectronic devices in mobile and wireless applications, the threat from variations in operating conditions, which is typically caused by supply voltage droops and temperature fluctuations, is continuing increasing. The most immediate manifestation of such variations is the delay uncertainty which can prevent circuits from meeting their timing specification, resulting in *timing errors*. Without proper protection, such timing errors can pose great threats to the reliability of digital systems. Currently, circuit designers typically combat timing errors by adding safety margins to the voltage and/or the clock frequency, known as guardbands. This practice leads to overly conservative design as the margins often exceed 40% of the nominal target specifications [11].

To avoid such pessimistic design while also protecting circuits from timing errors, designers must understand and model the timing effects of dynamic variations. Many models are proposed to predict timing errors of arithmetic instructions [16], [18], [4] or functional units (FUs) [22], [17], [21], [8]. For example, instruction-level models [16], [18], [4] predict timing errors based on instruction types. B-hive [22] and HFG [17] both use machine learning methods to predict timing errors of FUs under dynamic variations. Recently, to enable aggressive energy saving via voltage scaling, the research community embraces *approximate computing* by allowing errors only in computation units that can tolerate them [21], [8], [19]. This requires an accurate error models to explore the effects of timing errors at the application level. Existing error models include *single bit flip* [21], *bit flip with uniform probability* [8], and *last value* [19].

Unfortunately, despite significant prior efforts, variation-induced timing errors still cannot be accurately modeled and, subsequently, exposed to the application level for a holistic evaluation. This is due largely to a lack of consideration of workload variation that can completely change the manifestation of dynamic variation in timing errors. Specifically, while variations in operating conditions can change the delay of (critical) paths and hence the static delay of a circuit, this delay is not the "real" circuit delay at a certain time. The "real" (dynamic) circuit delay is the delay of the sensitized longest path, and path sensitization behavior is actually determined by circuit workload (Sec. III). To capture such dynamic delay in predicting timing errors, we propose an error model by jointly considering workload variations and variations in operating conditions. Specifically, our contributions are as follows:

- We perform dynamic timing analysis (DTA) to characterize circuit dynamic delay under a wide range of workload and operating conditions, based on which we collect training data. The DTA is based on extensive gate-level simulations with timing information extracted from a standard ASIC flow that considers physical details of post-layout designs in TSMC 45nm.
- We extract useful features from training data and apply supervised learning methods to build **TEVoT** that can predict timing errors of functional units (FUs) under different clock speeds, input workload, voltage, and temperature conditions. To the best of our knowledge, **TEVoT** is the first error model that can jointly consider the workload variations and variations in operating conditions.
- We expose circuit-level errors to application level for a holistic evaluation, based on which **TEVoT** can estimate application quality under different operating conditions.
- We evaluate **TEVoT** across 100 different operating conditions, 4 widely-used FUs, 3 clocking speeds, and 3 datasets. Results show that **TEVoT** achieves an average



(a) Initial state (b) First input changes (delay = $2ns$) (c) Second input changes (delay = $1.5ns$)

Fig. 1. Different delay under different input

prediction accuracy of 98.25% in predicting timing errors, and an average prediction accuracy of 97% in estimating application quality.

II. Related Work

To combat microelectronic variability, researchers proposed better-than-worst-case (BTWC) design methods [7] to enable guardband reduction, which, however, can induce considerable hardware overheads and performance penalty due to error detection and correction.

A less-intrusive way to combat variability is to model the timing errors in advance and then adaptively change the clock speed to improve efficiency. Various models are proposed to predict timing errors of arithmetic instructions [16], [18], [4] or functional units (FUs) [22], [17], [21], [8]. Instruction-level models predict timing errors based on the maximum delay of each instruction measured during simulation [16], [18], [4]. This method considers only instruction type to discriminate between instructions. Going down to the circuit level, B-Hive [22] divides timing errors into five categories and classifies them using decision trees; HFG [17] uses linear discriminant analysis to predict variability-induced errors of functional units. MACACO [24] uses Monte-Carlo simulation to model timing errors of voltage-scaled adders and multipliers. VARIUS [20] uses probabilistic analysis to model timing errors of microarchitectural blocks. Error models of voltage-scaled FUs are also intensively used in approximate computing research to assess the application-level effects. For example, *single bit flip* model flips one randomly chosen bit [21]; *bit flip with uniform probability* model flips different bits uniformly with a per-FU error probability [8], [13]. However, none of these works considers the impact of workload variations in predicting timing errors.

Main Difference: Our analysis and results indicate that input workload can completely change the manifestation of variability effects in timing errors. Thus, **TEVoT** is different than all the previous work as it jointly considers the workload variations and variations in operating conditions. Instead of statistical analysis, **TEVoT** can deterministically classify each circuit output as one of two classes: $\{$*timing correct*, *timing erroneous*$\}$, for a given input workload, clock speed, voltage, and temperature condition.

III. Problem Formulation

Timing error(s) of a given FU is a function f_e of clock speed, input workload, and operating condition as shown in Eq. 1. (We focus on dynamic variations here and but the same principle can be used to incorporate process and aging variations).

$$O = f_e(V, T, t_{clk}, I) \tag{1}$$

where V, T represent respective voltage and temperature condition, t_{clk} is the clock period, I is the input workload to the circuit, and O is the output class of an output value, $O \in \{\textit{timing correct, timing erroneous}\}$. Thus, a straightforward way of predicting timing errors is to learn the function of f_e directly. However, such direct prediction of timing errors lacks flexibility because once the circuit changes clock speed, the model needs to perform another inference.

Thus, we propose to predict circuit delay instead of predicting timing errors directly. This is because timing errors only occur when the clock period does not meet the circuit delay [3]. Circuits have two types of delays: static delay and dynamic delay. Static delay refers to the delay of the critical path in the circuit, which can be affected by variations of operating conditions. This delay, however, is not useful in our prediction because the critical path may not get sensitized. Actually, critical path is rarely sensitized by real-world workload [4]. In order to predict timing errors, we need to compare circuit clock period with the sensitized dynamic delay. The dynamic delay is the delay of the sensitized longest path in the circuit, which is determined by the input workload. For example, as illustrated in Fig. 1, the dynamic delay in Fig. 1 (b) is $2ns$ while the delay in Fig. 1 (c) is $1.5ns$. Thus, we represent the dynamic delay as a function f_d of input workload and operating conditions, as shown in Eq. 2.

$$D = f_d(V, T, I) \tag{2}$$

Once a delay is predicted, such delay can be reused to predict timing errors across different clock speeds. With this formulation, **TEVoT** can provide more flexibility and scalability.

Our goal is to learn (an approximation of) f_d given a set of inputs and operating conditions without any knowledge of the circuit structure. However, the potential input space of workload is huge. For a circuit with two 32-bit inputs, the potential input space is 2^{64}. Therefore, we propose to evaluate a set of supervised learning methods to classify the inputs.

IV. **TEVoT** Model

TEVoT is comprised of three phases as shown in Fig. 2: *Dynamic Timing Analysis, Model Training* and *Model Evaluation*. a) The *Dynamic Timing Analysis* phase implements the standard ASIC flow and uses gate-level simulation to generate dynamic delay under different input workload and operating conditions. b) In the *Model Training* phase, we extract useful features from training data and apply supervised learning

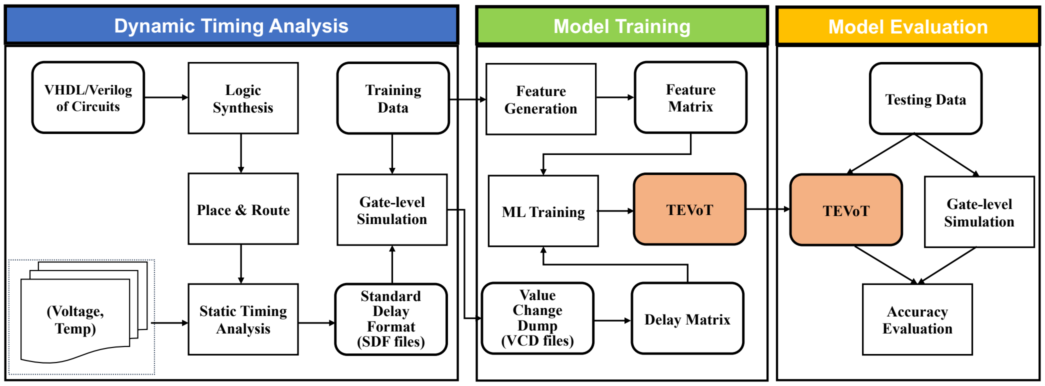


Fig. 2. The construction and evaluation of **TEVoT**.

methods to train **TEVoT**. c) In the *Model Evaluation* phase, **TEVoT** predicts timing errors for a given condition, which are compared with the simulation-based ground truth to evaluate prediction accuracy. More details about the three phases are illustrated as follows.

A. Dynamic Timing Analysis

The purpose of *Dynamic Timing Analysis* (DTA) phase is to generate dynamic delay under different workload and operating conditions. We focus on four FUs, 32-bit integer adder (INT_ADD) and multiplier (INT_MUL), and 32-bit floating point adder (FP_ADD) and multiplier (FP_MUL). These four most-widely used FUs are basic computation blocks for applications such as image-processing and deep learning applications and have been main modeling targets of similar works [22], [17]. The floating point units (FPUs) are compatible with IEEE-754 standard, and can provide more complex circuit structures compared to their integer counterparts.

We perform logic synthesis and place&route to generate gate-level netlists. Then, we perform static timing analysis (STA) to generate standard delay format (SDF) files under different operating conditions. To inject the dynamic variations, i.e., different voltage and temperature conditions, we use voltage-temperature scaling features of EDA tools to enable the composite current source method for modeling cell behavior. Thus, we generate an SDF file for each voltage-temperature (V, T) pair. We use 20 different voltage points and 5 different temperature points as shown in Table I (in total 100 different (V, T) pairs).

TABLE I
OPERATING CONDITION PARAMETERS.

	Start Point	End Point	Step	of Points
Voltage	0.81V	1.00V	0.01V	20
Temperature	0°	100°	25°	5
Clock Speedups	5%	15%	5%	3

We then feed SDF files to gate-level simulation to perform back-annotation simulation. Since the main purpose of gate-level simulation is to generate the value change dump (VCD) file for DTA, we perform simulation with a relatively slow clock period to make sure there are no timing errors. VCD files contain the switching activity of interested circuit nodes, e.g., output bits, in the circuit. With such detailed switching activities, we then compute the dynamic delay at each cycle. To get a dynamic delay at some cycle N, we use the time of the very last toggled event at the input pins of all sequential elements t' to subtract the arrival time of the positive clock edge t. For example, t' could be the time of the last toggled output bit. That is, the dynamic delay at this cycle is $t' - t$. We develop a Python script that can automatically parse VCD files to extract dynamic delay at each cycle.

B. Model Training

1) Feature Generation: The purpose of *Feature Generation* is to extract “useful variability feature” from raw training data. The training input workload comes from two sources: random data and application data profiled from real world applications. To cover a wide range of input space, we use the homogeneous distribution of two operands over 2D input space [22].

As presented in Eq. 2, dynamic delay is a function of operating conditions V, T and workload I. Operating conditions are typically at limited discrete levels such as shown in Table I, hence it is possible to train the model using all operating conditions. However, the potential input space of the workload is huge. For a circuit with two 32-bit inputs, the potential input space is 2^{64}. It is not feasible to apply all 2^{64} input patterns for training. Therefore, the key accomplishment of **TEVoT** is to predict timing errors under unseen input data by learning the path sensitization behavior under unseen input data.

We convert all input data to bit-level vectors since circuits receive binary input; since each bit affects path sensitization, each bit value is an individual feature. Next, we need to determine if we need to incorporate history inputs. To do this, we perform a path sensitization analysis. According to [2], 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 as shown in Fig. 1, both the previous and current input have impact on the path sensitization. For example, when the second input changes, whether the value of a node toggled depends on its current state, which is set by previous input. Thus, the previous input sets the state and current input toggles the circuit nodes based on current state. Thus, we include current input $x[t]$ (concatenation of multiple inputs), and history input $x[t-1]$ as our feature. We verify this conclusion by performing simulation with $100K$ cycles. For every 20 cycles, if we randomly vary the preceding input $x[t-1]$ while fixing current input $x[t]$, $D[t]$ varies irregularly; if we fix both $x[t-1]$ and $x[t]$, $D[t]$ is also fixed.

Therefore, our variability feature is $\{V, T, x[t], x[t-1]\}$ and output label is $D[t]$, which can then be used to predict timing errors across different clock periods. After *Feature Generation*, we can generate the following feature matrix I and delay matrix D, where $x[t] \in \{0,1\}^K$, and V and T are real numbers. For 32-bit circuit, $K = 64$ because $x[t]$ and $x[t-1]$ each has 64 bits. This makes a feature dimension of 130 and the possible input feature space is more than 2^{130}.

$$I = \begin{bmatrix} x[t] & x[t-1] & V^1 & T^1 \\ x[t+1] & x[t] & V^2 & T^2 \\ \vdots & \vdots & \vdots & \vdots \\ x[t+N] & x[t+N-1] & V^N & T^N \end{bmatrix} D = \begin{bmatrix} D[t] \\ D[t+1] \\ \vdots \\ D[t+N] \end{bmatrix} \tag{3}$$

2) ML Training: While specific classes of circuits show certain positive learnability results [14], they do not cover the circuits we consider here. In contrast, we focus on learning when a circuit does not work as desired, i.e., the circuit contains timing errors. Capturing the timing errors requires learning the dynamic path sensitization by specific input workload. Thus, we evaluate several widely-used supervised learning classification methods: k-nearest neighbor (**k-NN**), support vector machine (**SVM**), linear regression (**LR**), and random forest (**RF**) for their increased sophistication and practical use.

k-NN provides useful theoretical properties [5] and has limited parameters to train. **k-NN** predicts the target by local interpolation of the targets associated of the K nearest neighbors in the training set. **LR** and **SVM** can learn weights w on each feature including each bit position. By using these two methods, we consider the disparity of significance of different bit positions in sensitizing paths. **RF** is an ensemble learning method that constructs multiple decision trees and uses majority votes to improve accuracy and prevent overfitting. Decision trees are a non-parametric supervised learning method that aims to establish a set of decision rules from training data. This method emphasizes the disparity of different features as well as considering the interaction between different features.

Table.II presents the prediction accuracy, training and testing time of four methods using 200K training data and 200K test data under a computer configuration of 2-core Intel(R) Xeon(R) CPU E5504@2.00GHz and 50GB memory. Based on the results, we choose **RF** due to its high accuracy, fast computing time and superior interpretability. Actually, **RF** fits our task scenario better than other methods because it can interpret the significance disparity between different features (compared with KNN) and it considers the interactions among different bits/features (compared with SVM and LR). Since the training process is a one-shot offline activity, the testing time is more important for users. We will open-source the pre-trained models for research community.

TABLE II
PREDICTION ACCURACY, TRAINING AND TESTING TIME.

method	Accuracy	Training Time	Testing Time
LR	82.3%	6.84s	2.24s
KNN	81.7%	127s	3548s
SVM	92.2%	15653s	9879s
RFC	98.3%	142s	3.5s

C. Model Evaluation

For a given input I, voltage V, temperature T, and clock speed t_{clk}, **TEVoT** classify the corresponding output as either $correct$ or $erroneous$. We evaluate the model performance using prediction accuracy, and compare it with baseline models. The prediction accuracy is obtained by comparing **TEVoT** predicted result with simulation results:

$$prediction\_accuracy = \frac{\#matched\_cycles}{\#total\_cycles} \tag{4}$$

where $\#total\_cycles$ is the number of total simulation cycles, and $\#matched\_cycles$ is the number of cycles at which predicted result matched simulation result, i.e., both results are either C_c or C_e.

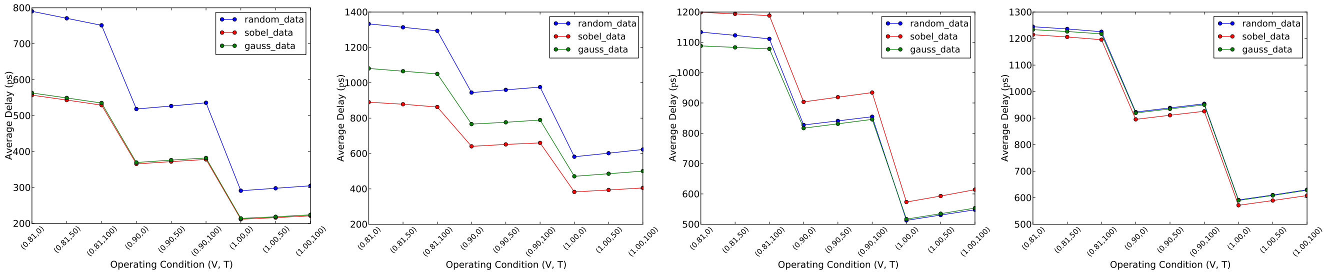
We compare **TEVoT** against following baseline models which can help us evaluate the true performance of our model:

- **Delay-based:** this model is from [16], [4], [17] where a timing error is predicted if the clock period does not meet the maximum delay measured offline at each operating condition. This model does not consider input workload but only instruction types, V, and T.
- **TER-based:** this model is from [19], [8] where a timing error is predicted with a probability based on the timing error rate (TER) measured during offline simulation. This model is widely-used in approximate computing.
- **TEVoT-NH:** this model is trained similarly with **TEVoT** except it does not consider computation history, i.e., preceding input $x[t-1]$, as input features.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

We generate the RTL descriptions of FUs using FloPoCo [6]. We peform logic synthesis, place&route, and STA using Synopsysis Design Compiler, IC Compiler and PrimeTime, respectively with TSMC 45nm technology. We peform gate-level simulation using Mentor Graphics ModelSim. We use three datasets, random dataset and real-world datasets profiled from two image processing applications from AMD APP SDK v2.5 [1], Sobel filter and Gaussian filter. The images are from butterfly image dataset in Caltech-101 [9]. For training, we use $200K$ randomly generated data and 5% randomly-picked images as training data; for testing, we use $200K$ (unseen) random data and the rest images as testing data. We profile application datasets by simulating the OpenCL codes of these applications with customized *Multi2Sim* [23] simulator, a cycle-accurate CPU-GPU heterogeneous architectural simulator. We also perform error injection, i.e., inject timing errors back to applications, using *Multi2Sim* to obtain application-level quality. We use 100 operating conditions as shown in Table I, and for each operating condition, we use



(a) INT_ADD delay

(b) INT_MUL delay

(c) FP_ADD delay

(d) FP_MUL delay

Fig. 3. Average delay under different datasets and operating conditions

TABLE III
Average Timing error prediction accuracy of **TEVoT** across 100 operating conditions and 3 clock speeds.

FU	random data				sobel data				gauss data			
	TEVoT	**Delay-based**	**TER-based**	**TEVoT**-NH	**TEVoT**	**Delay-based**	**TER-based**	**TEVoT**-NH	**TEVoT**	**Delay-based**	**TER-based**	**TEVoT**-NH
INT_ADD	99.9%	0.02%	96.5%	91.9%	99.2%	0.77%	82.0%	86.7%	99.7%	0.01%	65.7%	82.5%
FP_ADD	98.6%	7.6%	87.1%	89.9%	98.2%	16.3%	80.2%	83.4%	98.6%	3.2%	78.1%	91.5%
INT_MUL	97.1%	21.1%	73.7%	83.7%	99.2%	0.39%	17.7%	31.4%	99.5%	6.2%	79.8%	69.7%
FP_MUL	97.0%	10.2%	84.9%	87.5%	95.4%	9.4%	73.1%	80.1%	96.7%	11.4%	82.1%	85.4%

three clock speedups (5%, 10%, and 15%) from its fastest error-free clock frequency (so that the output has timing errors). We adopt machine learning methods from Scikit-learn [15] with default hyperparameter (e.g., 10 trees, all features considered during split for **RF**).

B. Delay Variations

We first characterize the dynamic delay variations under different operating conditions and datasets. Specifically, for each dataset and operating condition, we compute the average dynamic delay across the entire dataset. Fig. 3 presents an example of 9 (V, T) pairs ($V \in \{0.81, 0.90, 1.00\}$ and $T \in \{0, 50, 100\}$) and 3 datasets, based on which we can observe several important facts. First, the dynamic delay is varied with different operating conditions. Specifically, as voltage increases, the delay is reduced. However, the impact of temperature on delay is not fixed. For example, when the voltage is 0.81V, the increased temperature reduces delay; when the voltage is 0.90V, the increased temperature increases the delay. This phenomenon is known as the inverse temperature dependence [25]. Second, the delay can also be changed dramatically by different datasets. For example, for INT_ADD, the average delay of random dataset is 30% greater than that of application datasets. This indicates the significant impact of input workload on dynamic delay, thus motivating our consideration of the workload variations in delay modeling.

C. Timing Error Prediction

Table III presents **TEVoT** prediction accuracy against the baseline models: **Delay-based**, **TER-based**, and **TEVoT**-NH. We compute the average prediction accuracy of such models across 100 operating conditions and 3 clock speeds as shown in Table I. We can observe several important facts. First, for all the datasets and FUs, **TEVoT** can exhibit prediction accuracy beyond 95%. On average, **TEVoT** exhibits 98.25% prediction accuracy. As a comparison, **Delay-based** exhibits extremely low prediction accuracy (7.21% on average) because of its pessimistic prediction: it always predicts timing error when the clock period does not meet its measured maximum delay. This means that whenever there is a clock speed up, it predicts timing errors, ignoring the case that the input workload may sensitize smaller delay. The **TER-based** model achieves on average 75.07% accuracy. **TER-based** model uses a pre-determined error probability from training data to predict testing data, without using any information from testing data. However, the delay/error statistics of training data and testing data may deviate significantly. Lastly, **TEVoT**-NH presents an average accuracy of 80.30%. The difference between **TEVoT** and **TEVoT**-NH indicates the importance of incorporating the history input workload. Further, **TEVoT** is 100X faster than gate-level simulation on average across different FUs. Typically, the more complex the circuit structure is, the slower the simulation is. But for **TEVoT**, because it is based on a fixed set of decision rules, it will not scale up with the complexity of the circuit.

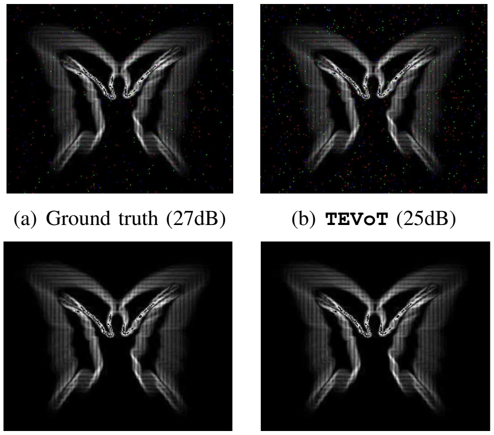
D. Application Quality Estimation

TABLE IV
Application quality estimation accuracy using four models.

Application	**TEVoT**	**Delay-based**	**TER-based**	**TEVoT**-NH
Sobel	97.6%	75.7%	53.8%	58.8%
Gauss	96.5%	84.1%	64.6%	71.2%

We present a case study of using **TEVoT** to estimate the application output quality under different operating conditions. Specifically, for each output image of Sobel filter and Gaussian filter, **TEVoT** can classify it into either *acceptable* (PSNR ≥30dB) or *unacceptable*. This is especially important in approximate computing for exploring quality-energy tradeoff.

At each operating condition and clock speed, we use gate-level simulation, **TEVoT**, **Delay-based**, **TER-based**, and **TEVoT**-NH to derive the corresponding timing error rates (TERs) of FUs. Then, we inject timing errors based on these TERs to applications using *Multi2Sim* simulator. During the



(a) Ground truth (27dB) (b) **TEVoT** (25dB)

(c) **TEVoT**-NH (56dB) (d) **TER-based** (48dB)

Fig. 4. Sobel filter output based on simulation (ground truth), **TEVoT**, **TEVoT**-NH, and **TER-based** models. (The noisy pixels are more visible on electronic version or color printing). Note that we do not put **Delay-based** model here because it always leads to completely corrupted output.

error injection process, we let the FUs return a random value each time they have timing errors, similar to [12].

$$estimation\_accuracy = \frac{\#matched\_estimations}{\#total\_estimations} \tag{5}$$

We use Eq. 5 to compute the estimation accuracy. **TEVoT** presents on average 97% estimation accuracy, while the baseline models present 79.9%, 59.1%, and 65% average accuracy. Specifically, **Delay-based** would always estimate the output quality as $unacceptable$ because it always predicts timing errors when there is clock speedup. This prediction may be consistent with actual outputs. For example, Fig. 4 shows an $unacceptable$ output ($27dB$) of Sobel filter: **TEVoT** estimates correctly because its output is $25dB$; **TER-based** and **TEVoT**-NH are incorrect because their estimations are $acceptable$.

E. Discussion

Usage: **TEVoT** can help circuit designers perform early design space exploration; software developers can assess their program resilience to hardware variations without access/knowledge to circuit simulation.

Scope: We focus on arithmetic circuits as they often represent timing-critical parts in a pipeline [26], [4] and they are widely-used in approximate computing [22], [17], [19]. Our future work will incorporate other circuit types such as memory.

Learning method: While the selection and tuning of learning algorithm is important to achieve good accuracy, it is not the main focus of this paper. We leave this direction open to follow up research, e.g., applying more advanced learning algorithms.

VI. Conclusion

We propose **TEVoT**, a supervised learning model that can predict the timing errors of functional units under variations in operating conditions and workload. We perform extensive dynamic delay characterization under a wide range of operating conditions and extract useful features from the input data to predict the dynamic delay, based on which we can predict timing errors across different clock speeds. We apply random forest methods to train **TEVoT**. On average across 100 operating conditions, 3 clock speeds, 4 functional units, and 3 datasets, **TEVoT** can obtain an average prediction accuracy at 98.25%, significantly higher than baseline models. We further use **TEVoT** to estimate the application quality for two image-processing applications and **TEVoT** can estimate the quality with an 97% accuracy. Our future work focuses on developing error models for more variation parameters such as process variations and apply them on other circuit types.

References

[1] Amd app sdk v2.5. [online]. available: http://www.amd.com/stream.

[2] Michael Bushnell et al. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Springer Science & Business Media, 2004.

[3] Hari Cherupalli et al. Graph-based dynamic analysis: Efficient characterization of dynamic timing and activity distributions. In *ICCAD*, 2015.

[4] Jeremy Constantin et al. Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment. In *DATE*, 2015.

[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 et al. Designing custom arithmetic data paths with flopoco. *IEEE Design & Test of Computers*, 2011.

[7] Dan Ernst et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *MICRO*, 2003.

[8] Hadi Esmaeilzadeh et al. Architecture support for disciplined approximate programming. *ASPLOS*, 2012.

[9] Li Fei-Fei et al. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer vision and Image understanding*, 2007.

[10] Puneet Gupta et al. Underdesigned and opportunistic computing in presence of hardware variability. *TCAD*, 2012.

[11] Rajesh K Gupta et al. Variability expeditions: A retrospective. *IEEE Design & Test*, 2019.

[12] Xun Jiao et al. An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations. In *ICCAD*, 2017.

[13] Evgeni Krimer et al. Lane decoupling for improving the timing-error resiliency of wide-simd architectures. *ISCA*, 2012.

[14] Nathan Linial et al. Constant depth circuits, fourier transform, and learnability. *Journal of the ACM (JACM)*, 1993.

[15] Fabian Pedregosa et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 2011.

[16] Abbas Rahimi et al. Analysis of instruction-level vulnerability to dynamic voltage and temperature variations. In *DATE*, 2012.

[17] Abbas Rahimi et al. Hierarchically focused guardbanding: an adaptive approach to mitigate pvt variations and aging. In *DATE*. IEEE, 2013.

[18] Abbas Rahimi et al. Application-adaptive guardbanding to mitigate static and dynamic variability. *Computers, IEEE Transactions on*, 2014.

[19] Adrian Sampson et al. Enerj: Approximate data types for safe and general low-power computation. In *PLDI*, 2011.

[20] Smruti R Sarangi et al. Varius: A model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing*, 2008.

[21] John Sartori et al. Stochastic computing: embracing errors in architectureand design of processors and applications. In *CASES*, 2011.

[22] G Tziantzioulis et al. b-hive: A bit-level history-based error model with value correlation for voltage-scaled integer and floating point units. In *DAC*, 2015.

[23] Rafael Ubal et al. Multi2Sim: A Simulation Framework for CPU-GPU Computing . In *PACT*, 2012.

[24] Rangharajan Venkatesan et al. Macaco: Modeling and analysis of circuits for approximate computing. In *ICCAD*, 2011.

[25] Sean H Wu et al. How does inverse temperature dependence affect timing sign-off. In *Emerging Technologies and Circuits*. Springer, 2010.

[26] Jing Xin et al. Identifying and predicting timing-critical instructions to boost timing speculation. In *MICRO*, 2011.