AVATAR: An Aging- and Variation-Aware Dynamic Timing Analyzer for Error-Efficient Computing

Zuodong Zhang, Zizheng Guo, Yibo Lin, *Member, IEEE*, Meng Li, *Member, IEEE*, Runsheng Wang, *Member, IEEE*, Ru Huang, *Fellow, IEEE*

***Abstract*—As the timing guardband consumes more and more design margin with the technology scaling, better-than-worst-case (BTWC) techniques have gained more attention as a promising solution. BTWC techniques can relax the design margin by transcending the pessimistic static timing constraints and utilizing the dynamic timing information. However, to guarantee the design reliability throughout the lifetime, the conventional dynamic timing analysis (DTA) engines need an extra reliability guardband, which is commonly evaluated under the worst-case corners of aging and variation. This type of guardbanding consumes the precious design margin, thus hindering the efficiency improvement from BTWC techniques. Therefore, in this paper, we propose AVATAR, an aging- and variation-aware dynamic timing analyzer that can perform DTA with the impact of transistor aging and random process variation, including the gate-level aging analysis and random variation model that can accurately calculate cell delay under the impact of transistor aging and random variation, and an event-based DTA algorithm that avoids the pessimistic property of graph-based analysis. We also propose an ML-assisted DTA acceleration flow for the multi-cycle DTA of homogeneous multicore designs. We present two case studies using AVATAR to show its effectiveness. First, we present an application-based dynamic-voltage-frequency-scaling (DVFS) design methodology based on AVATAR, which can exploit application-level dynamic timing slack (DTS) to improve energy efficiency and performance. The results demonstrate that, compared to the design based on the conventional corner-based DTA, the additional performance improvement of the design based on AVATAR can be up to 14% or the additional power-saving can be up to 20%. Second, we demonstrate using the proposed ML-assisted acceleration flow for reliability-aware deep neural network (DNN) accelerator simulation. We use the proposed flow to estimate the impact of timing errors due to aging and random variation on the inference accuracy of two benchmark DNNs. The results demonstrate that the proposed acceleration flow achieves up to 10× speedup with an average error of less than 2%.**

***Index Terms*—Dynamic timing analysis (DTA), aging-aware timing analysis, random variation, machine learning (ML), timing error evaluation**

The preliminary version, entitled "AVATAR: An Aging- and Variation-Aware Dynamic Timing Analyzer for Application-based DVAFS", has been presented at the Design Automation Conference (DAC) in 2022. This work was supported in part by the National Key R&D Program (2020YFB2205500), NSFC (62125401, 62141404, 62034007) and the 111 Project (B18001).

Z. Zhang, Z. Guo are with the School of Integrated Circuits, Peking University, Beijing, China.

Y. Lin, R. Wang and R. Huang are with the School of Integrated Circuits, Peking University, Institute of Electronic Design Automation, Peking University, Wuxi, China, and Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China.

M. Li is with the School of Integrated Circuits, Peking University, and the Institute for Artificial Intelligence, Peking University, Beijing, China.

Corresponding author: Yibo Lin (yibolin@pku.edu.cn) and Meng Li (meng.li@pku.edu.cn).

I. Introduction

As the CMOS technology scales to nanoscale, the design margin has become extremely tight due to reliability and manufacturability issues like transistor aging and random process variation (local variation) [1]. To guarantee the product's yield and the circuit functionality during the expected lifetime, designers usually choose to add timing or voltage guardbands. However, the guardbands are estimated by static timing analysis (STA) in the worst-case corners, and increase rapidly with CMOS scaling, which may eventually obliterate gains from device scaling [2], [3].

To overcome the above dilemmas, better-than-worst-case (BTWC) techniques are proposed to further improve performance or efficiency. BTWC methods usually transcend the pessimistic static timing constraints, and adopt optimization techniques such as dynamic-voltage-frequency-scaling (DVFS) [4], [5], [6] and timing speculation [7], [8], [9]. On the other hand, many emerging algorithms have nondeterministic results and are inherently error-tolerant at the algorithm level. some BTWC techniques further propose to trade precision, accuracy, or even reliability for performance or energy efficiency, such as approximate computing and application-level error-tolerant design [10], [11], [12], [13]. All these techniques depend on the dynamic timing information calculated by dynamic timing analysis (DTA).

Moreover, with the ever-increasing demands of image processing and machine learning (ML) applications, domain-specific architectures (DSA) such as deep neural network (DNN) accelerators have been widely deployed from data centers to edge devices. BTWC techniques are widely used in DNN accelerator design because of the high demand for energy efficiency and inherent error tolerance of ML algorithms. For example, many recent works focus on exploiting inherent error tolerance [14], [15], [16] or timing speculation [17], [7], [8], [9] to improve the energy efficiency of DNN accelerators. All these works also need DTA to evaluate the effectiveness and explore the most optimal design parameters at the design phase.

However, the conventional DTA is implemented by first performing a gate-level simulation with delay annotation, then using a post-processing program to extract the dynamic delay and analyze the toggle paths for each cycle [5], [10]. This implementation has two drawbacks. First, the net delay and cell delay used during the gate-level simulation are dumped from graph-based STA, which struggles with the pessimistic properties of the graph-based analysis [18], [19], [20]. Sec-

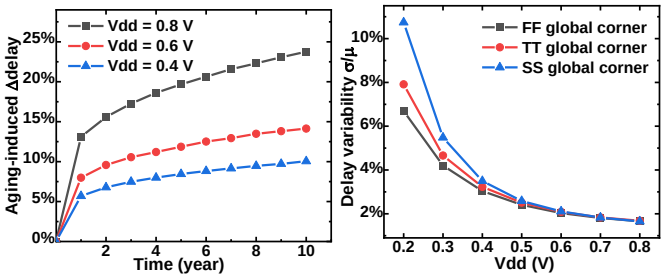
ondly, the conventional DTA tools need extra reliability guardbands in timing analysis, and the guardbands are estimated under the worst-case aging and random variation. The margin estimated by the corner-based method often exceeds 40% of the nominal target specifications [3]. The pessimism stemming from the dynamic delay calculation and reliability guardbands will ultimately make the BTWC design an over-designed system.

Therefore, in this paper, we propose AVATAR, an aging- and variation-aware dynamic timing analyzer. AVATAR can accurately analyze the dynamic timing information under the impact of transistor aging and random variation. We also propose an ML-assisted DTA flow that can effectively accelerate multi-cycle DTA with negligible errors. To the best of our knowledge, AVATAR is the first DTA algorithm that enables gate-level aging and random variation analysis. The novel contributions of this paper are summarized as follows:

1) We propose AVATAR, an aging- and variation-aware dynamic timing analyzer, which can estimate the impact of transistor aging and random variation on path delay calculation in DTA. Compared to the conventional corner-based analysis, AVATAR supports accurate dynamic delay and delay variability estimation under a specific aging scenario, which avoids pessimistic guardband.
2) We develop an event-based DTA algorithm, which simulates the switching activities based on event propagation for each cycle. The proposed DTA algorithm can calculate the dynamic delay with accurate timing information, and support the gate-level aging and variation model.
3) We propose a new ML-assisted acceleration flow to speed up the homogeneous multi-core systems (especially DNN accelerator) simulation. We develop an ML classifier that can filter out input patterns that are less likely to cause timing errors, thereby significantly reducing the simulation cycles while maintaining high accuracy.
4) We present how AVATAR can be applied to guide the application-based DVFS strategy design, which is an effective optimization technique that can improve energy efficiency by enabling application-based voltage/frequency adjustment.

We evaluate the proposed application-based DVFS strategy on an open RISC-V core and a set of embedded application benchmarks. The results demonstrate that, compared to the design based on the conventional corner-based DTA, the additional performance boosting of the proposed application-based DVFS strategy based on AVATAR can be up to 14% or the additional power-saving can be up to 20%. We also use the proposed ML-assisted flow to estimate the classification accuracy of two benchmark DNNs after aging. The results show that the proposed flow can speed up the DTA up to 10 with an error of less than 2%.

The rest of this paper is organized as follows. In Section II, the preliminary of reliability issues in advanced technology nodes, static timing analysis, and dynamic analysis are presented, and we formulate the problem. In Section III, the implementation details of the aging-&variation-aware DTA algorithm are presented. Section IV describes the ML-assisted acceleration methods. Section V presents the use case of AVATAR in application-based DVFS. Section VI presents the experiment results of the ML-assisted DTA flow in accelerator simulation. Finally, Section VII concludes the paper.



(a) (b)

Fig. 1: (a) The delay increase of FO4 delay because of the impact of transistor aging at different V_{dd}. (b) The delay variability of FO4 delay which is normalized to the delay at nominal V_{dd}. All the data in this figure is collected from HSPICE transient simulation with a 16/14 nm FinFET modelcard.

II. Preliminaries

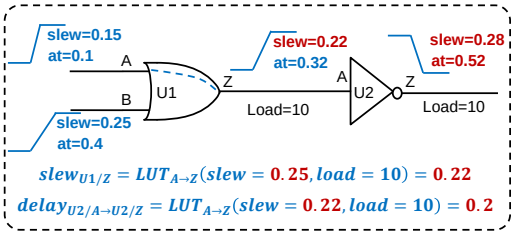
A. Reliability Concerns in Timing Analysis

The aggressive scaling of transistor size makes the transistor aging and local process variation more and more pronounced in the nanoscale, which threatens the correct system functionality. Transistor aging effects reduce the driving capability of the transistors, which increases the circuit delay and the probability of timing errors over time. While random process variation causes path delay variation, increasing the complexity of timing analysis, especially in near-threshold voltage (NTV). Fig. 1 shows the cell delay increase due to aging and random variation. The cell used is a Fan-out-of-4 (FO4) inverter, which is always used when optimizing critical paths. The results show that, in the nominal voltage, the delay variability is mainly caused by the aging effect. While the impact of random variation on cell delay becomes more significant when performing aggressive voltage scaling.

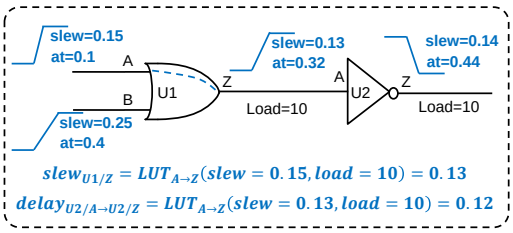
To avoid the timing errors due to random variation and transistor aging that may impact the functionality and the yield, designers typically resort to guardbanding, which adds an extra margin in timing analysis [2]. However, the guardband of aging and random variation is estimated under the worst-case corner. Moreover, the guardband is increasing rapidly due to technology scaling. As a result, more performance, power, and area (PPA) gains are being sacrificed for guardband, which may eventually obliterate PPA benefits from technology scaling. Therefore, aging- and variation-aware design flow is urgently needed.

B. Static Timing Analysis

STA can find the longest timing path on topology and calculate the critical path delay. Depending on the imple-



(a)



(b)

Fig. 2: The delay calculation in (a) the GB-STA is pessimistic because of the input slew merge. While the delay calculation in (b) the PB-STA is accurate.

mentation algorithm, STA can be divided into the graph-based STA (GB-STA) and the path-based STA (PB-STA). The GB-STA constructs a directed acyclic graph (DAG) based on the circuit netlist, which consists of nodes (the input and output ports and cell pins) and edges (the timing arcs of cells and nets). GB-STA calculates the critical delay in two steps: forward propagation, and backward propagation. In the forward propagation step, the timing information of each timing arc is estimated, which includes delay, slew, and arrive time (*at*). In the backward propagation step, GB-STA calculates the required arrive time and finds the top-K paths.

GB-STA can traverse the whole graph and calculates the critical delay quickly [21]. However, the delay calculation in GB-STA is pessimistic because it propagates the worst-case slew. Fig. 2(a) gives an example of slew propagation and delay calculation in GB-STA. In the setup check, the slew calculation of each cell uses the worst slew of all input pins. For example, the slew calculation of the pin $U1{=}Z$ will utilize the worst slew, i.e. slew of pin $U1{=}B$, thus it would be overestimated. Moreover, the delay calculation of the gates in the next level ($U2$) will also be estimated.

Compare to the GB-STA, PB-STA supports more accurate delay calculation. Because PB-STA is used for path delay calculation of particular paths, and the slew and delay calculation utilizes the realistic input slew(Fig. 2(b)) [20], [19]. For example, when the path $fU1{=}A\ !\ U1{=}Z\ !\ U2{=}A\ !\ U2{=}Zg$ is analyzed in PB-STA, PB-STA tools would calculate the slew of $U1{=}Z$ based on the input slew of pin $U1{=}A$ (Fig. 2(b)). Such input slew merge accounts for the most pessimism in path delay calculation of GB-STA. Although PB-STA is more accurate than GB-STA, it is too time-consuming, as we cannot enumerate all paths in a large circuit. In practice, designers tend to utilize GB-STA first to find the top-K critical paths, and then recalculate these paths in PB-STA mode. In advanced technology nodes, increasing design complexity and clock frequency leads to more and more expensive PB-STA.

C. Dynamic Timing Analysis

STA is a widely used timing analysis algorithm for decades, but it can not calculate dynamic delay according to realistic inputs and evaluate the timing error rate under certain workloads. While BTWC techniques usually need dynamic information such as the dynamic timing slack (DTS) and toggle rates to relax the pessimistic design constraints.

Graph-based DTA (GB-DTA) is a commonly used technique to calculate such dynamic information [5], [10]. GB-DTA is usually implemented by the delay-annotated gate-level simulation with a post-processing program, and the cell delay used is extracted from GB-STA. There are two main disadvantages of the conventional GB-DTA, the first is the pessimistic path delay calculation due to the previously mentioned worst slew merge. And the second is that the timing information calculated by these tools requires extra aging and random variation guardbands calculated under the worst-case corner to cover the impact of non-ideal factors. Such pessimism in DTA will eventually lead to the over-design of the BTWC system.

D. Problem Formulation for Aging- and Variation-aware Dynamic Timing Analysis

DTA is usually utilized to estimate the dynamic delay for specific input vectors. We define the problem of aging- & variation-aware DTA for a single cycle as follows:

Problem. Given a circuit netlist, a specific input vector, and the specific aging workload, simulate the switching activities of each internal node with timing information, report the dynamic delay distribution of each timing endpoint under the impact of transistor aging and random variation.

Definition 1. *Event:* a digital switching signal on a pin. The *at* (arrive time) of an *event* is the time of signal switching relative to the last clock. The *slew* of an *event* is the transition time of the signal switching.
Definition 2. *Triggered pin:* a pin is triggered in a particular cycle if the value of the pin changes in that cycle.
Definition 3. *Triggered path:* a path is triggered in a particular cycle if all the pins in the path are triggered in that cycle.
Definition 4. *Aging workload:* Typical workload of a circuit over its expected lifetime, which is used to calculate aging after a specific aging time.

In practice, the multi-cycle DTA can be implemented by performing the single-cycle DTA repeatedly.

III. Event-based Aging- and Variation-aware DTA

In this section, the implementation details of the proposed aging- and variation-aware DTA algorithm are presented. The workflow of AVATAR is depicted in Fig. 3. It includes two parts: the gate-level aging and variation analysis, and the event-based DTA.

A. Gate-level Aging and Variation Model Characterization

The gate-level aging and variation analysis require the corresponding cell-level information, which is modeled as an

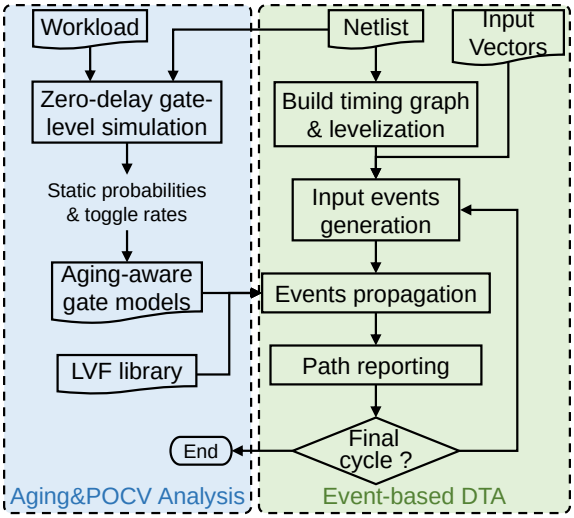


Fig. 3: Overview of the task flow for AVATAR.

aging-aware timing model and a variation-aware timing model. The aging-aware timing model is characterized by the method proposed in [22], in which the impact of transistor aging on the cell *delay* and *transition time (tr)* is modeled as follows:

$$delay_{aged} = delay_{fresh} + \sum_{i \in I} a_i \quad V_{thi} \tag{1}$$

$$tr_{aged} = tr_{fresh} + \sum_{i \in I} b_i \quad V_{thi} \tag{2}$$

where I is the set of all transistors in the cell, $delay_{fresh}$ and tr_{fresh} are obtained from the liberty library. According to the SPICE simulation, we found that the sensitivity coefficients $a_i; b_i$ also depend on the slew of the input signal and the load of the output pin. Therefore, we use the first-order linear model is used to fit the dependency as follows:

$$a_i = a_{i0} + \quad _{ai} \quad slew + \quad _{ai} \quad load \tag{3}$$

The polynomial ridge regression is employed to fit the equations above.

In the characterized gate-level aging model, the inputs for querying an aged cell *delay* or transition time are the corresponding timing arc, the input slew on the input pin, the output load, the V_{th} of transistors, and operating conditions(such as supply voltage and temperature). And the outputs of the gate-level aging model are the cell *delay* and *tr* after aging.

For random process variation analysis, the parametric on-chip variation (POCV) analysis [23] is adopted. POCV can be implemented by the liberty variation format (LVF) library, a model that has been proven and widely used in commercial libraries. LVF library captures the cell delay variability and transition time variability due to random variation at multiple slew and load combinations in look-up tables. To reduce the complexity of characterization and analysis, random process variations are modeled by assuming that it is independent of the transistor aging [24], and we only model the increase of the means value of cell *delay* and *tr* due to transistor aging. It is noted that the model characterization requires lots of SPICE simulations, but these simulations only need to be executed once for each standard cell library.

B. Workload Analysis

The workload analysis is for providing the necessary information for aging analysis, i.e. the switching activities (duty factor and toggle rate) of each gate by simulating a realistic working scenario. As mentioned earlier, the necessary inputs of the gate-level aging-aware timing model are the V_{th} of each transistor in a standard cell. Therefore, the expected results of workload analysis are the V_{th} of each transistor. Since transistor aging is mainly caused by negative bias temperature instability (NBTI) [25] in digital circuits, this work only calculates the V_{th} degradation of PMOS.

We perform workload analysis in two steps: first, we use zero-delay gate-level simulation to estimate the switching activity (toggle rate and static probability) of internal nets; then, we employ a cell-level analytical model [22] to estimate the stress of each transistor, and the long-term aging model for calculating the V_{th}. It is noted that the zero-delay gate-level simulation can be replaced by the delay-annotated gate-level simulation or the event-based DTA, which is more accurate, but the runtime also increases substantially. After collecting the V_{th} information, combined with the liberty library, the event-based DTA engine can query the timing information of any cell after aging.

C. Event-based DTA

The DTA engine in AVATAR is an event-based algorithm. In the event-based algorithm, the aforementioned gate-level aging-aware model and the switching activities of workload analysis are adopted to query the aged cell *delay* and *tr* after aging. To support random variation analysis, deterministic timing information (including delays, arrival times, slacks, etc.) is replaced with a Gaussian distribution, which can be represented by two parameters, the mean value ( ) of the distribution and the stand deviation ( ). Hence, the dynamic delays in the timing report of AVATAR are also distributions.

In the event-based algorithm, an event refers to the signal switching of an internal net or a pin. Each event has at least three basic attributes: signal type (rising or falling), *slew*, and *arrival time (at)*. Different from the conventional delay-annotated graph-based DTA, in the event-based algorithm, each event has its own *slew* property, and the input slew used for the cell delay calculation is the accurate slew. Therefore, the path delay calculation in the event-based DTA algorithm can avoid graph-based pessimism, and achieve the same accuracy as the path-based analysis.

As shown in Fig. 3, the first task in the event-based DTA is reading the input netlist and building the timing graph. Then, we levelize the timing graph, this step is to build the level-by-level dependencies of nets and gates for the subsequent events propagation algorithm.

The next task is the cycle-by-cycle logic simulation and dynamic timing analysis. In each cycle, the task can be divided into three sub-steps: input event generation, event propagation, and path reporting.

Fig. 4: The events propagation process on a simple circuit. For a brief illustration of the event propagation algorithm, we ignore the wire delay and set all cell delays as 0.1.

Fig. 5: An example of the event propagation for cell FA.

1) Input Event Generation:AVATAR first reads the input vector, and compares the bit-wise value of two adjacent cycles. A changed value means a digital switching occurs, so we need to build an input event on the corresponding pin.

We set the slew of input events as the user-defined input_slew, and we set the at of the input event as the sum of the user-defined external_delay and a random offset, defined as input_uncertainty. The input_uncertainty is used for the following two reasons: (1) Realistic input signals are not perfectly aligned because of the clock skew or jitter of the previous stage flip-flops or process and voltage variations. (2) A random offset ensures that multiple input events will not arrive at exactly the same time on any gate. This helps to eliminate the ambiguity in the propagation algorithm.

2) Event Propagation:In this step, the input events are propagated to the corresponding timing endpoints (e.g. data pin of flip-flops or the output pin of the netlist). Fig. 4 shows an example to describe how events are propagated in the timing graph. The events are propagated through gates and nets alternately. Moreover, the propagation algorithm through a net and the propagation algorithm through a gate are different.

For a net in a circuit netlist, there is one input pin and at least one output node. Propagation through a net will only increase the at of events by net delay. Therefore, we access each output pin in turn and add output events to it. The at of the output event is the at of the input event plus the corresponding wire delay.

While the event propagation algorithm through gates is more complicated, because the new event generation requires logic simulation and timing calculation. Moreover, for these multi-input gates, the algorithm has to deal with the case that there are several input events from different pins.

Algorithm 1 shows the propagation algorithm through a gate. We first collect all the arrival times of the input events and sort them in chronological order (lines 1-3). Then, we perform the logic simulation of the gate at each time and obtain the corresponding output value (lines 5-6). A changed value on the output pin in two adjacent times means that there is an output event. For each output event, the algorithm first determines the triggered timing arc and query the and of the cell delay and the output tr from the characterized aging-aware model and the LVF library (lines 8-10). Finally, the algorithm calculates the at and slew of the output event according to library information and the input event that triggers the current output event (lines 11-14), and add the output event to the event list of the output pin (line 15). After completing all output events, the algorithm traverses the output event list and removes the improper event pairs, which include the collision case or the event pairs that the arrive time interval is less than the transition time (line 16). Because there will be no complete signal switching for these cases.

Fig. 5 presents an example to illustrate the propagation process and the corresponding delay calculation. In the current cycle, there are two input events on the input pin A and B. We collect the arrive time of these input events and get the sorted list of time f0:2; 0:6g. Then, we perform the logic simulation of the Full Adder (FA) cell at each time point and obtain the output value. The value of output pin CO does not change, while the value of pin S changes twice. Therefore, there are two output events on pin S triggered by the two input events. Next, the corresponding delay (both and), and output tr after aging are queried from the characterized gate-level aging

Algorithm 1: Propagate events through the gate.

```
Input: gate g_i, event list of input pins
Output: event list of output pins
build a list of time T(g_i);
T(g_i) = get_input_event_arrive_time(g_i);
sort T(g_i) and ensure that there are no two input
  events with the same at;
for all time t_i ∈ T(g_i) do
    find the corresponding input event event_in;
    get the states of input pins at time t_i, and calculate
      the output value;
    for all pin p_i ∈ PIN_out whose state changed do
        // calculate cell delay and tr
        delay_aged = LUT(event_in.slew, load) +
          aging_model(event_in.slew, load, ΔV_th);
        delay_fresh = LUT(event_in.slew, load);
        tr_aged = LUT(event_in.slew, load) +
          aging_model(event_in.slew, load, ΔV_th);
        // build a new event
        build an output event event_out;
        event_out.at = event_in.at + delay_aged;
        event_out.at = Sqrt((event_in.at)^2 + (delay_fresh)^2);
        event_out.slew = tr_aged;
        add event_out to the events list of p_i;
filter the improper event pairs
```

model and LVF library. Finally, we calculate the output events according to the cell delay and input events.

3) Path Reporting: After all event propagation is complete, the algorithm reports the dynamic delay and the longest triggered path. AVATAR iterates through the event lists of all timing endpoints, and finds the latest event. Then (and) are reported as the critical dynamic delay distribution of the current cycle. For path reporting, AVATAR starts from the latest event, and finds the input event that triggered this output event at the previous stage, and reports the corresponding input pin. Repeat reporting pins until it is a primary input pin. After reporting the dynamic delay and path, AVATAR clears all the event lists and reserves the final state of internal pins in the current cycle, which are recorded as the initial logic state for the event propagation of the next inputs. This operation guarantees that AVATAR does not take up too much memory as the simulation cycle increases.

D. Parallel Acceleration

In practice, the main factor causing the prohibitive runtime of DTA is not the size of the circuit, but the excessive number of simulation cycles (usually > 100M) [11]. Therefore, we adopt a cycle parallel scheme to accelerate the simulation and timing analysis. The cycles to be analyzed are divided into n equal parts, which are assigned to different CPU threads for independent execution. Fig. 6 presents the rules for task assignment. It should be noted that, there should be no cycle overlap when splitting the input vectors, and the overlap size m is determined by the pipeline depth, because internal nodes require m cycles to be in the logic state that it is supposed to be in. Compared with the netlist splitting scheme, the proposed cycle parallelism avoids additional segmentation algorithms, and different subtasks can be executed independently. These advantages allow for better scalability of AVATAR.

Fig. 6: The task assignment scheme for parallel execution in AVATAR.

IV. ML-ASSISTED DTA FLOW

Although DTA is the most accurate pre-silicon method in capturing the real impact of timing errors, it is not widely used due to the prohibitively long simulation time. Therefore, in addition to developing faster parallel DTA tools, recent works propose to adopt ML algorithms to further accelerate the DTA [26], [27], [28], [29]. As discussed in the previous section, the dynamic delay is determined by the input pattern and operation conditions (including but not limited to temperature, voltage, and aging). Several previous works proposed to use ML classifiers to predict timing errors according to the input patterns [26], [29]. Nevertheless, to predict the timing error under multiple clock periods and operation conditions, designers have to develop multiple models for each case. Ma et al [27] propose to use regression models to directly predict the dynamic delay under different input patterns, voltages, and temperatures.

However, these algorithms cannot be applied to accelerate the proposed aging- and variation-aware DTA because of the following reasons: 1) The gate-level aging-induced delay depends on the switching activity of each gate. Extra input features are needed to enable aged dynamic delay prediction, which will increase the number of input features by several orders of magnitude. 2) Generating training data with many input patterns under a wide range of operating conditions is very time-consuming. Also, a large training set can make the model training very slow, thus effective acceleration may not be achieved. 3) Although the average prediction error is acceptable, it is difficult for these ML models to precisely predict the extreme values (shown in section VI-A). Thus the estimation error is large when predicting small TER.

To address the aforementioned challenges, we divide the factors affecting dynamic delay into two parts: 1) The input pattern, determine which paths will be triggered. 2) The operation conditions that affect the path delay of these triggered paths. Hence the problem of predicting the dynamic delay after aging is divided into two problems: predicting the paths triggered by the specific input pattern and accurately

calculating the aged path delay. Such a problem decomposition is based on the observation that only a small percentage of input patterns can cause timing errors, which are defined as critical input patterns. We propose an ML model as a pre-filter to identify the critical patterns from a large number of inputs and only adopt AVATAR to simulate the critical patterns. As we will demonstrate in the next section, the pre-filter can significantly reduce the number of cycles to be simulated, thus achieving several times acceleration.

Fig. 7 shows the overall workflow of the proposed ML-assisted DTA. The proposed ML-assisted DTA flow can be used for all homogeneous multi-core systems, and we use the DNN accelerator simulation as an example. Note that the Model Training phase is being executed only once for developing the ML model. We will demonstrate the proposed flow in the context of the systolic array-based accelerator with the output stationary dataflow, which is used in the Google TPU [30] and is a commonly used architecture for DNN acceleration.

A. Input Extraction

In the first step, we offload the quantized DNN into the determined size systolic array. Then, we perform a behavior simulation of the specific systolic dataflow to obtain the input vectors and golden output vectors of MACs. We only consider the systolic dataflows that use local communication, such as output stationary, weight stationary, and input stationary.

B. Model Training

1) Input Feature.: The previous works have demonstrated that, besides the current input vectors, the input vectors of the previous cycles affect timing errors. And the window of history effect is equal to the pipeline depth [31], [29] of the circuit. Because the dynamic delay depends on the previous state of the circuit and the next state it is about to switch to, the state of an m-stage pipeline circuit is determined by the previous m inputs. Therefore, the input features are $\{x[t], \ldots, x[t-d+1], y\_golden[t], \ldots, y\_golden[t-d+1]\}$, where d is the pipeline depth.

2) Labels.: To assign labels to the training data, we use AVATAR to calculate the corresponding fresh dynamic delays of each timing endpoint. As long as the dynamic delay of any endpoint is greater than 80% of the critical path delay, we label the current input pattern as the critical pattern. In other words, we define the input patterns that will trigger the first 20% of the long path as critical input patterns. The reason why the delay threshold is taken as 80% of the critical path delay is that the clock period is determined according to the critical path, and the derate factor to cover the effect of PVTA variation is generally greater than 0.8. It should be noted that the delay threshold can also be freely defined according to the actual design scenario.

3) Training Process.: Previous works have explored multiple supervised ML algorithms to predict timing errors, and their results all indicate that the random forest classifier (RFC) is more competitive in terms of prediction accuracy and model complexity [27], [29]. Therefore, we adopt a balanced random forest classifier from the imbalanced-learn framework of Python [32], because the dataset is highly biased (the trigger rate of the long timing paths is low). We keep most of the hyperparameters as default, and only set `min_samples_split` to 6 to reduce the size of the model and prevent over-fitting.

C. Aging-aware Accelerator Simulation

The proposed aging-aware accelerator simulation is implemented in two steps: TER estimation and error-injection simulation. We first adopt the ML-assisted DTA flow to estimate the per-layer bit-wise TERs. We should note the TER of each output bit is estimated separately, because different endpoints are located on different timing paths. In this step, the trained ML model is deployed to identify the critical patterns from the original input vectors. Because the critical patterns usually account for a small percentage, the number of cycles to be simulated can be greatly reduced, thus achieving several times of acceleration.

Next, we calculate the bit error rates (BERs) of the output activations based on the TERs, and use error-injection simulation to estimate the classification accuracy. The BER of the output activations in the output stationary dataflow is calculated as:

$$BER(i) = 1 - \prod^{N} (1 - TER(i)) \quad (4)$$

where N is the number of MAC operations required for one convolution in the corresponding layer. The error-injection simulation is implemented using PyTorch. We randomly flip the corresponding bit of output activations (before the activation function) according to its BERs. Generally speaking, in most arithmetic units, only a few MSBs have non-zero BERs, which means the error injection does not increase runtime significantly. To avoid randomness error, the `batch_size` is set to 128 and the error-injection simulation is repeated five times with different seeds for each BER combination.

V. Application-based DVFS

We target the application-based DVFS as one of the use cases of AVATAR. Application-based DVFS is a widely adopted optimization method in embedded systems, which can effectively improve energy efficiency [4], [5], [6]. The motivation is based on the experimental observation that specific instructions and operands may not trigger all functionalities when being executed on an embedded core. Therefore, there will be a slack between the longest path delay calculated by STA and the largest delay of the paths triggered by the current instruction, which is defined as the DTS of the instruction. Similarly, the timing slack between the critical path of STA and the longest path that may be triggered by an application is defined as the DTS of the application. The DTS of a specific application means that the V_{min} could have been smaller or the f_{max} could have been larger than the nominal value when the processor executes the application. Therefore, allocating an optimal V_{min} /f_{max} for each application instead of using the fixed V_{dd}/f_{max} can improve the efficiency or performance of an embedded processor.

Fig. 7: Work ow of the proposed ML-assisted DTA for accelerator simulation.

TABLE I: Performance improvement from application-based DVFS based on the corner-based DTA and AVATAR. The processor operates at the nominal V_{dd} of 0.8V.

Benchmark [33]	Corner-based DTA [4], [5]			AVATAR (this work)				
	Delay (ps)	Max Freq. (MHz)	Impro. (vs STA)	Delay (ps)		Max Freq. (MHz)	Impro. (vs STA)	Additional Bene ts (vs corner-based DTA)
SHA	1054.95	948	13.75%	957.65	7.65	1020	22.38%	8.63%
AES_CBC	1132.18	883	5.99%	1028.41	7.76	951	14.10%	8.11%
FIR	1092.71	915	9.82%	991.29	7.56	986	18.35%	8.53%
BubbleSort	772.28	1290	55.38%	701.84	7.94	1380	65.36%	9.98%
Motion_Detection	1043.48	958	15.00%	945.27	7.56	1030	23.97%	8.97%
CNN	1151.86	868	4.18%	1045.53	7.69	936	12.30%	8.12%
Convolution	1151.78	868	4.19%	1045.70	7.69	936	12.28%	8.09%
2d_Filter	1068.33	936	12.33%	927.48	7.37	1050	26.37%	14.04%
MatrixMult	1092.05	916	9.89%	991.29	6.75	989	18.63%	8.74%
DCT	852.47	1170	40.77%	768.55	6.72	1270	52.15%	11.38%

Fig. 8: Overview of tool ows for the application-based DVFS.

The DVFS design ow is demonstrated in Fig. 8. We rst compile the embedded applications to executable the binary les of the target processor, and then we execute these applications in the RTL simulation. The aim of the RTL simulation is to get the corresponding input vectors of the timing critical functional units (FUs). It is worth noting that AVATAR can directly perform the timing analysis for the full processor, but this work only calculates the dynamic delay of the FUs in the processor because the previous work [5] has shown that the critical paths are mainly located in the FUs. Moreover, timing errors in the FUs will only result in wrong computing value and will not cause the crash of the application, which indicates that some timing errors can be tolerated in some applications [29].

Then, the RTL description of the target processor is synthesized into the gate-level netlists. We use AVATAR as the DTA engine to calculate the dynamic delays of the timing critical FUs. The reports of AVATAR include the dynamic delays of each timing endpoint and the longest path triggered per cycle, by which we can also estimate the timing error rate (TER) of each application. Finally, we allocate the optimal V_{min} / f_{max} for each benchmark application based on the DTS and TER.

A. Experimental Setup

We implement AVATAR in C++ and perform all the experiments on a Linux machine with Intel Xeon E5-2650 at 2.20GHz and 64 GB RAM. AVATAR is executed in parallel with 32 CPU threads by default. The benchmark embedded processor is RI5CY, an open-source RISC-V core [33], and the benchmark applications include cryptographic kernels (SHA and AES_CBC), control-intensive applications (BubbleSort), transformation (DCT), and data-intensive linear algebra kernels (FIR, CNN, 2d_Filter, and MatrixMult). We use Synopsys Design Compiler [34] to synthesize the RISC-V core with the open-source Nangate 15nm standard cell library [35]. We

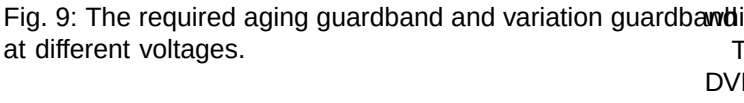


Fig. 9: The required aging guardband and variation guardband at different voltages.

perform static timing analysis and generate sdf files by using Synopsys Primetime [36]. The Power under different voltages is also evaluated by Primetime PX [37]. The LVF libraries are characterized at each V_{dd} between 0.8V and 0.4V at 0.02V interval with a commercial 16/14nm FinFET modelcard, using Siliconsmart [38]. Gate-level aging-aware delay models are characterized by the NBTI model from [25], which has been calibrated with the silicon data of 16/14 nm FinFETs. The maximum frequency of the implemented processor is 833MHz at the nominal voltage 0.8V.

We use the following two methods to perform DTA and determine the application-specific V_{min} / f_{max}:

1) Using the corner-based DTA with extra aging guardband and variation guardbands in dynamic delay calculation [4], [5]. Therefore, the reported dynamic delays are calculated as $delay * (1 + total\_guardband)$. According to the previous works, the aging guardband is assumed to be 15% [22] and the random variation guardband is assumed to be 5% [39] at nominal V_{dd}. And the trend of the guardband with different voltages is estimated by the FO4 delay [40], as shown in Fig. 9. As the supply voltage V_{dd} decreases, the needed total guardband first decreases because of the abatement of the aging effect. But as the V_{dd} continues to decrease to the near threshold region, the total guardband begins to gradually increase because of the increasing impact of random variation.
2) Using AVATAR. The dynamic delay reported by AVATAR already includes the impacts of aging and variation, and thus, no extra guardband is needed. The final delay is calculated as $(delay) + 3\ (delay)$.

To avoid the errors arising from the small amount of input data, for each application, the number of simulation cycles is at least 200k cycles and at most 2M cycles. By comparing the performance and energy efficiency improvement of these two methods, we show how AVATAR enables better DVFS at runtime.

B. Performance and Power Improvement

Table I shows the performance improvement by adopting the performance-first DVFS strategy based on the corner-based DTA and AVATAR. It can be seen that the maximum dynamic delay of different applications is different, depending on which logic paths can be triggered. For example, the maximum dynamic delays of the application CNN, AES_CBC are close to the static critical path delay, because these applications contain lots of multiplications, thus may trigger the critical paths of the FU multiplier. While the maximum dynamic delays of the application BubbleSort, DCT are significantly less than the static critical path delay, because the instructions in the application BubbleSort are mainly executed by the comparator unit in the ALU. As for the application DCT, it also contains lots of multiplications, but the operands are 8-bit integers, which will not trigger the long logic paths of the FU multiplier.

Table I also shows that, compared to the static design, the DVFS strategy based on the corner-based DTA can improve the performance by an average of 17.13%, while the DVFS strategy based on AVATAR can improve the performance by an average of 26.59%. The additional improvement of AVATAR over the corner-based DTA varies in different applications. Overall, AVATAR's additional performance improvement is 8% to 14%. This is because the longest paths triggered by different applications are different, but the total guardband is the same for all paths in the corner-based DTA. In practice, however, the delay degradation and variability of a path depend on the type of cells on the path and the workload.

Table II shows the power saving by adopting the power-first DVFS strategy based on the corner-based DTA and AVATAR. On average, the DVFS strategy based on the corner-based DTA can save power by 38%, while the strategy based on AVATAR can save power by 50.57%. Compared with the corner-based DTA, AVATAR's additional power-saving can be up to 20%. It is worth noting that the minimum energy point is about 0.2V for the used technology, which is lower than the V_{min} in table II. Therefore, the strategy based on AVATAR not only saves more power, but also gains more energy efficiency.

VI. DNN Accelerator Simulation

We target DNN accelerator simulation as another use case of AVATAR and the proposed ML-assisted DTA flow.

The systolic array with the output stationary data flow is used as the benchmark DNN accelerator, which contains 65K (256 × 256) MAC units with 8-bit signed weights, 8-bit unsigned activations, and 24-bit partial sums. We synthesize the benchmark systolic array with the open-source Nangate 15nm standard cell library, using Synopsys Design Compiler. Two DNNs for the CIFAR-10 dataset are adopted (the parameters of each DNN benchmark are shown in Table III) to evaluate the accuracy.

In the Input Extraction step, the mapping and scheduling are implemented by the open-source tool SCALE-Sim [41]. And the quantization and re-training are implemented in the PyTorch framework.

We estimate the TER and the classification accuracy using the following methods:

Full-Sim: simulates all MAC operations using AVATAR. The reports of AVATAR are the bit-wise timing error rates after aging.

TABLE II: Power saving from application-based DVFS based on the corner-based DTA and AVATAR. The processor operates at the nominal frequency of 833MHz.

Benchmark [33]	Corner-based DTA [4], [5]			AVATAR (this work)				
	Min Vdd (V)	Delay (ps)	Power Savings	Min Vdd (V)	Delay (ps)		Power Savings	Additional Bene ts (vs corner-based DTA)
SHA	0.66	1177.19	38.00%	0.60	1144.04	11.44	50.73%	12.73%
AES_CBC	0.72	1191.76	23.25%	0.64	1165.70	10.45	42.41%	19.16%
FIR	0.68	1193.89	33.32%	0.62	1152.21	10.72	46.65%	13.33%
BubbleSort	0.48	1185.59	71.01%	0.46	1111.08	20.18	73.76%	2.75%
Motion_Detection	0.64	1193.09	42.41%	0.60	1130.30	11.30	50.73%	8.32%
CNN	0.74	1193.56	17.87%	0.66	1157.36	9.88	38.00%	20.13%
Convolution	0.74	1193.81	17.87%	0.66	1157.58	9.88	38.00%	20.13%
2d_Filter	0.66	1190.69	38.00%	0.60	1155.70	10.61	50.73%	12.73%
MatrixMult	0.68	1192.24	33.32%	0.62	1146.01	9.48	46.65%	13.33%
DCT	0.52	1186.73	64.92%	0.50	1103.88	14.49	68.04%	3.12%

TABLE III: The number of MAC operations to be simulated for a test set containing 128 inputs in different DNN architecture.

Name	DNN Architecture	MAC OPs
LeNet-5	L1-L2 (Conv): (32,32,3) (14,14,6) L3-L5 (FC): 400 120 84 10	80M
AlexNet	L1-L5 (Conv): (32,32,3) (7,7,96) (3,3,256) (3,3,384) (3,3,384) L6-L9 (FC): 2304 1024 512 10	6000M

ML-assisted uses the trained RFC model as a pre- lter to identify the critical input patterns, and only simulates the critical patterns using AVATAR. In our implemented systolic array, the pipeline depth of a single MAC unit is one, so our input feature is $x[t]; x[t-1]; y\_golden[t]; y\_golden[t-1]$.

DelayNet this model is from [28] where a fully-connected DNN with sigmoid activations is deployed to directly predict the dynamic delay according to the input vectors. To achieve bit-wise timing error rate estimation, we modi ed the number of output neurons to 4 (representing the delay of the most signi cant four bits), and increased the number of neurons in the hidden layer to 50 proportionally to the number of input neurons. Because the DelayNet can only predict fresh dynamic delay, an extra aging guardband is also needed. And the extra guardband is calculated by the same method mentioned in Section V-A.

The RFC model and DelayNet are trained with the same training data, which contains 3M input features extracted from different benchmark DNNs.

A. Accuracy Evaluation

Fig. 10 shows the TERs of the most signi cant four bits calculated by different methods. The other bits are ignored because there is no timing error. The input vectors are extracted from the second convolution layer in AlexNet. It can be seen that the TER calculated by ML-assisted ow is very close to that calculated by Full-Sim. The average error of TER estimation by ML- ow is only 1.5e-6. Although each path has a different aging rate, our method covers the top 20% of paths, so the RFC model still nds most of the inputs that will cause timing errors after aging. The TERs calculated by DelayNet are not plotted in Fig. 10 because it is one or two orders of magnitude different from the exact result. We should note that the mean absolute percentage error (MAPE) of the predicted result from DelayNet is reasonable, but predicting an error rate of less than 1e-4 is beyond the capability of the model. More speci cally, the prediction error of DelayNet is rather large in predicting the extreme values (delay closed to the critical path delay), so the TERs calculated by DelayNet are mostly above 1e-4.

Fig. 11 shows the classi cation accuracy on different benchmark DNNs. Similar to the results of TER estimation, the accuracy calculated by the proposed ML-assisted ow is very close to that calculated by Full-Sim. The sensitivity of classi cation accuracy to the clock period is also well captured by the proposed ow. While the classi cation accuracy estimated by DelayNet is lower than the accurate results because of its overestimation in layer-wise TERs. The detailed estimation errors of the proposed ow and DelayNet on different benchmark DNNs are listed in Table IV. The result shows that the estimation errors of the proposed ML-assisted ow are less than 2%, on both benchmark DNNs.

Meanwhile, the results also indicate that the inherent error tolerance of systolic array for timing errors is limited, and the timing errors due to aging can cause signi cant classi cation accuracy loss. For example, without an extra aging guardband, ten years of aging will reduce the classi cation accuracy of LeNet by 34% and will reduce the classi cation accuracy of AlexNet by 57%. The reason for the signi cant decrease in classi cation accuracy is mainly due to the fact that in modern medium-scale or large-scale CNN, one convolution operation requires hundreds or thousands of MAC operations. Even if the TER of a single MAC operation is relatively small, the BER of the output activations can be large. Our observations also coincide with the previous silicon testing data [17]. More proactive error prevention or correction strategies and HW/SW co-optimization are necessary to build robust accelerators.

B. Runtime Evaluation

Table IV shows the speedup of the proposed ML-assisted ow and DelayNet compared to Full-Sim. Fig. 12 shows the runtime breakdown of different methods on two benchmark DNNs. The ML inference and AVATAR are both executed with 32 CPU threads on a Linux machine with Intel Xeon

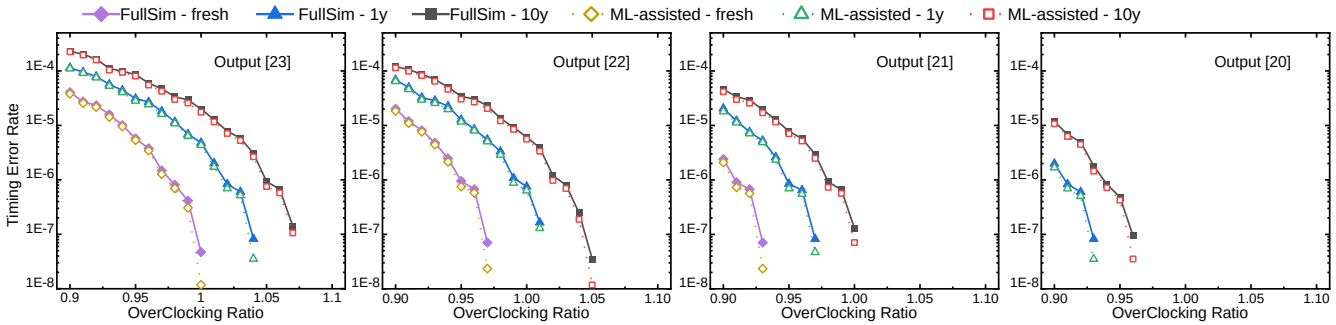


Fig. 10: TERs of different output bit positions for different aging times are calculated by different methods. The input vectors are extracted from the second convolution layer in AlexNet.

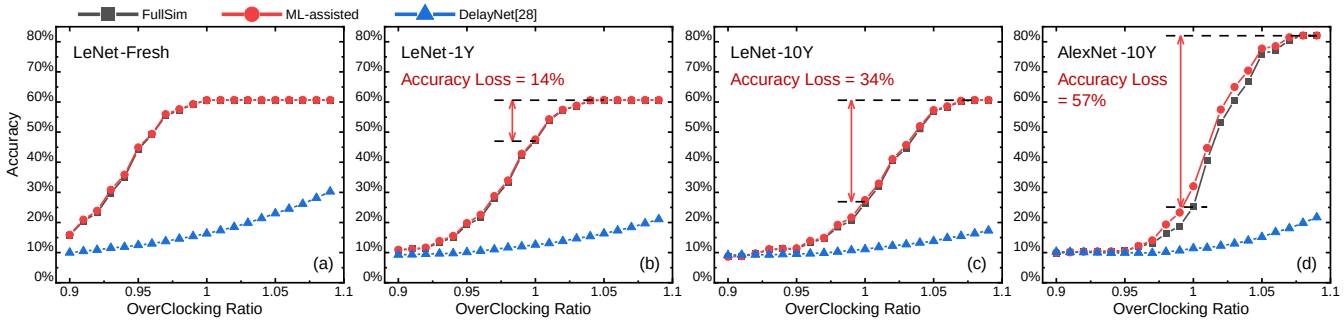


Fig. 11: Classification accuracy on different benchmark DNNs calculated by different methods.

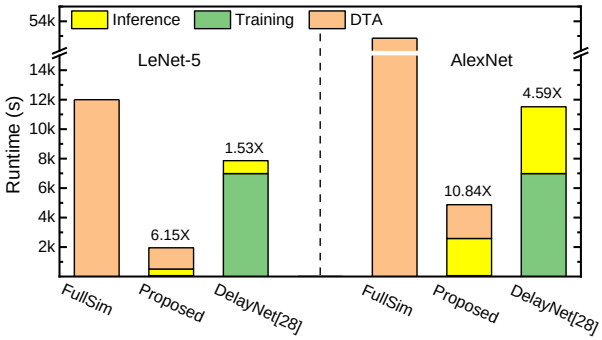


Fig. 12: Runtime breakdown of different simulation flow on benchmark DNNs.

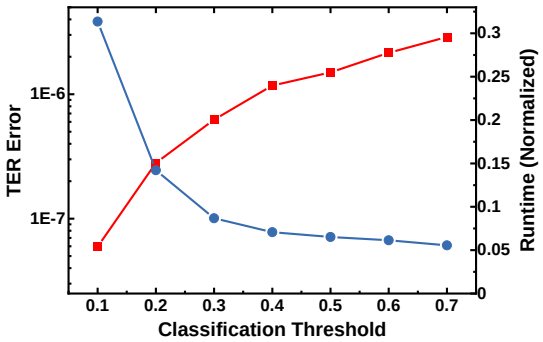


Fig. 13: Trade-off between accuracy and runtime by adjusting classification threshold in RFC model.

E5-2650 at 2.20GHz and 64 GB RAM. Even with the RFC model training time, the proposed ML-assisted flow is still 6-11 faster than Full-Sim. Because the critical patterns from the RFC model only account for 5-10% of the original inputs (the actual percentage is less than 5%). Moreover, the training process of the RFC model is very fast, which only consumes 63.35s. The speedup of DelayNet is worse than the proposed ML-assisted flow due to the long training time (6983.77s). If only care the evaluation phase, the runtime of DelayNet and the proposed ML-assisted flow are similar. However, as the scale of CNN increases, the sparsity of the convolution layers also increases, so the advantages of RFC will be more obvious.

We also demonstrate that the trade-off between accuracy and runtime can be achieved by adjusting the classification threshold in the RFC model. The default threshold is 0.5. As shown in Fig. 13, if the classification threshold is reduced, the number of inputs classified as the critical pattern will increase, and therefore the runtime of AVATAR will also increase. However, the missed inputs that actually trigger timing errors are reduced, thus improving the accuracy. Therefore, the designer can flexibly adjust the threshold value according to the design scenario.

VII. Conclusion

In this paper, we propose AVATAR, an aging- and variation-aware dynamic timing analyzer, which can calculate the dynamic delay with the impact of transistor aging and random variations. Compared to the conventional corner-based DTA, AVATAR can accurately estimate the impact of aging and variation on timing, thus avoiding pessimistic guardband. To evaluate its effectiveness, we use AVATAR to estimate the application-specific V_{min}/f_{max} in application-based DVFS de-

TABLE IV: The accuracy and speedup of the proposed ML-assisted flow and DelayNet.

benchmark	ML-assisted			DelayNet [28]		
	error	speedup w/ training	speedup w/o training	error	speedup w/ training	speedup w/o training
LeNet	0.43%	6.15X	6.36X	19.25%	1.53X	13.67X
AlexNet	1.83%	10.84X	10.99X	25.61%	4.59X	11.66X

sign. The results demonstrate that the additional performance improvement of the application-based DVFS flow based on AVATAR can be up to 14% or the additional power-saving can be up to 20%, compared with the conventional flow. We also propose an ML-assisted accelerator flow for DNN accelerator simulation. The proposed flow achieves up to 10 DTA speedup with an error of less than 2% in inference accuracy estimation of DNN accelerators.

REFERENCES

[1] R. Huang, X. Jiang, S. Guo, P. Ren, P. Hao, Z. Yu, Z. Zhang, Y. Wang, and R. Wang, "Variability-and reliability-aware design for 16/14nm and beyond technology," in *IEEE International Electron Devices Meeting (IEDM)*, 2017, pp. 12.4.1–12.4.4.

[2] A. Rahimi, L. Benini, and R. K. Gupta, "Variability mitigation in nanometer cmos integrated systems: A survey of techniques from circuits to software," *Proceedings of the IEEE*, vol. 104, no. 7, pp. 1410–1448, 2016.

[3] R. K. Gupta, S. Mitra, and P. Gupta, "Variability expeditions: A retrospective," *IEEE Design & Test*, vol. 36, no. 1, pp. 65–67, 2019.

[4] H. Cherupalli, R. Kumar, and J. Sartori, "Exploiting dynamic timing slack for energy efficiency in ultra-low-power embedded systems," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2016, p. 671–681.

[5] J. Constantin, L. Wang, G. Karakonstantis, A. Chattopadhyay, and A. Burg, "Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2015, pp. 381–386.

[6] T. Jia, R. Joseph, and J. Gu, "19.4 an adaptive clock management scheme exploiting instruction-based dynamic timing slack for a general-purpose graphics processor unit with deep pipeline and out-of-order execution," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2019, pp. 318–320.

[7] J. Zhang, K. Rangineni, Z. Ghodsi, and S. Garg, "Thundervolt: Enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *ACM/IEEE Design Automation Conference (DAC)*, 2018.

[8] P. Pandey, P. Basu, K. Chakraborty, and S. Roy, "Greentpu: Improving timing error resilience of a near-threshold tensor processing unit," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[9] N. D. Gundi, T. Shabanian, P. Basu, P. Pandey, S. Roy, and K. Chakraborty, "Effort: A comprehensive technique to tackle timing violations and improve energy efficiency of near-threshold tensor processing units," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 29, no. 10, pp. 1790–1799, 2021.

[10] I. Tsiokanos, L. Mukhanov, and G. Karakonstantis, "Low-power variation-aware cores based on dynamic data-dependent bitwidth truncation," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2019, pp. 698–703.

[11] O. Assare and R. Gupta, "Accurate estimation of program error rate for timing-speculative processors," in *ACM/IEEE Design Automation Conference (DAC)*, 2019.

[12] Z. Zhang, R. Wang, Z. Zhang, R. Huang, C. Meng, W. Qian, and Z. Zhou, "Reliability-enhanced circuit design flow based on approximate logic synthesis," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2020, p. 71–76.

[13] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Dvafs: Trading computational accuracy for energy through dynamic-voltage-accuracy-frequency-scaling," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2017, pp. 488–493.

[14] W. Choi, D. Shin, J. Park, and S. Ghosh, "Sensitivity based error resilient techniques for energy efficient deep neural network accelerators," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[15] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *ACM/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[16] X.-X. Wu, Y.-W. Hung, Y.-C. Chen, and S.-C. Chang, "Accuracy tolerant neural networks under aggressive power optimization," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2020, pp. 774–779.

[17] P. N. Whatmough, S. K. Lee, D. Brooks, and G.-Y. Wei, "Dnn engine: A 28-nm timing-error tolerant sparse deep neural network processor for iot applications," *IEEE Journal Solid-State Circuits*, vol. 53, no. 9, pp. 2722–2731, 2018.

[18] J. Bhasker and R. Chadha, *Static timing analysis for nanometer designs: A practical approach*. Springer Science & Business Media, 2009.

[19] Z. Guo, T.-W. Huang, and Y. Lin, "A provably good and practically efficient algorithm for common path pessimism removal in large designs," in *ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, December 2021.

[20] G. Guo, T.-W. Huang, Y. Lin, and M. Wong, "Gpu-accelerated critical path generation with path constraints," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Virtual Conference, November 2021.

[21] Z. Guo, T.-W. Huang, and Y. Lin, "Gpu-accelerated static timing analysis," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020, pp. 1–9.

[22] Z. Zhang, R. Wang, X. Shen, D. Wu, J. Zhang, Z. Zhang, J. Wang, and R. Huang, "Aging-aware gate-level modeling for circuit reliability analysis," *IEEE Transactions on Electron Devices (TED)*, vol. 68, no. 9, pp. 4201–4207, 2021.

[23] A. B. Kahng, "New game, new goal posts: A recent history of timing closure," in *ACM/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[24] B. Kaczer, T. Grasser, P. J. Roussel, J. Franco, R. Degraeve, L.-A. Ragnarsson, E. Simoen, G. Groeseneken, and H. Reisinger, "Origin of nbti variability in deeply scaled pfets," in *IEEE International Reliability Physics Symposium (IRPS)*, 2010, pp. 26–32.

[25] S. Guo, R. Wang, Z. Yu *et al.*, "Towards reliability-aware circuit design in nanoscale finfet technology: — new-generation aging model and circuit reliability simulator," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 780–785.

[26] X. Jiao, A. Rahimi, Y. Jiang, J. Wang, H. Fatemi, J. P. de Gyvez, and R. K. Gupta, "Clim: A cross-level workload-aware timing error prediction model for functional units," *IEEE Transactions on Computers*, vol. 67, no. 6, pp. 771–783, 2018.

[27] D. Ma, X. Zhang, K. Huang, Y. Jiang, W. Chang, and X. Jiao, "Devot: Dynamic delay modeling of functional units under voltage and temperature variations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 1–1, 2021.

[28] J. Zhang and S. Garg, "Fate: Fast and accurate timing error prediction framework for low power dnn accelerator design," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[29] S. Tompazi, I. Tsiokanos, J. M. Del Rincon, and G. Karakonstantis, "Estimating code vulnerability to timing errors via microarchitecture-aware machine learning," *IEEE Design & Test*, pp. 1–1, 2021.

[30] N. P. Jouppi, C. Young, N. Patil *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[31] I. Tsiokanos, L. Mukhanov, G. Georgakoudis, D. S. Nikolopoulos, and G. Karakonstantis, "Defcon: Generating and detecting failure-prone instruction sequences via stochastic search," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2020, pp. 1121–1126.

[32] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html