Learning-Oriented Reliability Improvement of Computing Systems From Transistor to Application Level

Behnaz Ranjbar*, Florian Klemme†, Paul R. Genssler†, Hussam Amrouch†,
Jinhyo Jung‖, Shail Dave‡, Hwisoo So‖, Kyongwoo Lee‖, Aviral Shrivastava‡, Ji-Yung Lin§¶,
Pieter Weckx§, Subrat Mishra§, Francky Catthoor§¶, Dwaipayan Biswas§, Akash Kumar*
* Chair of Processor Design, CFAED, Technische Universität Dresden, Dresden, Germany
† Chair for Semiconductor Test and Reliability (STAR), University of Stuttgart, Stuttgart, Germany
‖ Yonsei University, South Korea ‡ School of Computing and Augmented Intelligence, Arizona State University, USA
§ IMEC, Leuven, Belgium ¶ KU Leuven, Leuven, Belgium

***Abstract*—Due to technology scaling in modern computing platforms, the safety and reliability issues have increased tremendously, which often accelerate aging, lead to permanent faults, and cause unreliable execution of applications. Failure in some computing systems like avionics may cause catastrophic consequences. Therefore, managing reliability under all circumstances of stress and environmental changes is crucial in all abstraction layers, from application to transistor levels. Machine learning techniques are recently being employed for dynamic reliability estimation and optimization. They can adapt to varying workloads and system conditions. This paper presents reliability improvement approaches from multiple perspectives—from transistor-level to application-level—and discusses their effectiveness and limitations as well as open challenges.**

***Index Terms*—Aging, Cross-layer reliability, Device and circuit reliability, Dynamic reliability estimation, Error mitigation, Machine learning for systems, Task scheduling, Timing reliability.**

I. Introduction

TECHNOLOGY advancement has enabled computing systems to become an integral part of human life. However, the ongoing technology scaling is introducing an ever-increasing number of reliability challenges, especially when it comes to advanced technology nodes [1], [2]. This endangers the correct operation of hardware and software of computing processors. A failure of such systems (applications or hardware) may lead to catastrophic consequences. To design a reliable system, mitigation and countermeasures against error and aging need to be applied across multiple abstraction layers of the computing system, from transistors to all the way up to the application design, since various levels may be involved in the error and aging process [3]. Several design- and run-time approaches should be exploited, which can adapt to varying system conditions, execution requirements, and workloads variations during run-time. Machine learning (ML) techniques are recently employed for dynamic reliability improvement [4]. They can effectively adapt to such variations and determine effective system configuration under dynamic and environmental changes. Although ML techniques are very promising for improving reliability, they introduce several challenges in each and across layers that need to be carefully considered.

This paper discusses various aspects of learning-based reliability monitoring and improvement in computing systems.

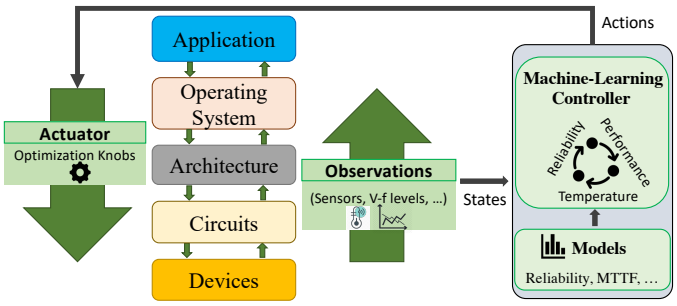

Fig. 1. Learning-based reliability management.

Fig. 1 shows a workflow for learning-based reliability management in which actions refer to optimization knobs used by the learning controller, states are the inputs based on the observation, and the agent aims to optimize the reward function, i.e., reliability improved in different abstraction layers, by using the resiliency models like mean time to failure (MTTF). In this paper, we first discuss how ML techniques can be used to estimate and mitigate aging from the transistor level to the standard-cell and circuit levels (Section II). Section III describes how ML can help to alleviate challenges of reliability modeling and improvement at the architectural level. Then, Section IV presents a survey of learning-based approaches that could improve reliability at the application layer through operating system (OS) techniques. We then analyze the reliability on a fault-tolerant system and discuss the timing costs in correcting register-level errors in Section V. Finally, we discuss the open challenges and limitations in Section VI and present summary in Section VII.

II. Estimating Device and Circuit Reliability Using Classical and Emerging Machine Learning

Reliability is one of colossal concerns for circuit designers. Transistor self-heating (SHE) is increasingly challenging because transistor scaling is reaching atomic levels in which quantum confinement becomes substantially prominent. With more confined 3D structures (e.g., TSMC Nanosheet FETs and Intel Ribbon FETs), heat arising in the transistor's channel cannot be easily dissipated and is hence "trapped" inside the transistor's channel. Such heat trapped inside the transistor

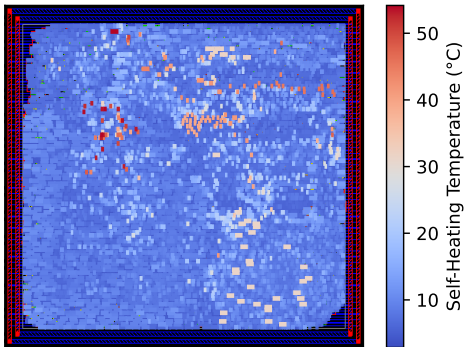

Fig. 2. Transistor self-heating temperatures within a RISC-V processor core. The shown temperatures are *above chip temperature*. Accurately modeling the temperatures for individual standard cells allows for more accurate and less pessimistic estimations of required safety margins [10].

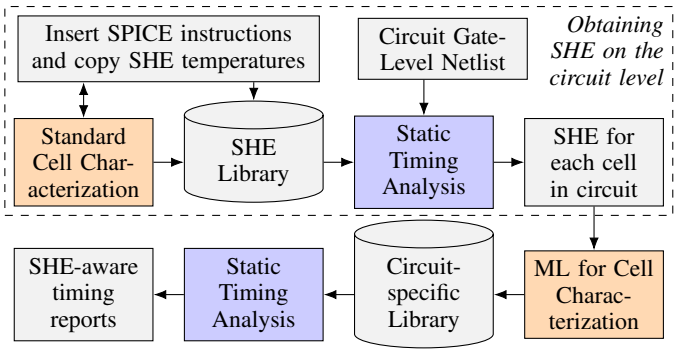

Fig. 3. Flow to obtain accurate circuit timings under the impact of transistor self-heating. The flow employs conventional Static Timing Analysis (blue) and specifically adjusted characterization flows (orange) for both SHE [10] and ML-based operation, enabling rapid generation of thousands of individual cells [9].

cannot be revealed (during the design time) even when using thermal setups that employ infrared cameras [5].

During runtime, this largely exacerbates the underlying aging mechanisms in transistors [6], which when combined with the existing challenged due to IR drops [7] can lead to serious reliability threats, especially in advanced technologies [8] At the design time, it is profoundly challenging to estimate safety margins that incur minimum overhead while preventing/mitigating implications of aging and self-heating during the entire projected lifetime. This is because foundries do not share their calibrated physics-based models that comprise of proprietary information about technology and material.

ML techniques (both classical and brain-inspired methods) can open new doors for foundries to train accurate models that empower circuit designers to estimate the actual impact of aging, from the material and transistor level to the circuit and processor level, without sharing any confidential physics-based models [9]. With certain adjustments to standard cell libraries, well-established EDA tools can be employed to upheave self-heating effects in individual devices at the transistor level and all the way up to entire processors/chip at the final layout level [10].

The challenge incurred by transistor SHE at the circuit level is showed in Fig. 2, in which all individual standard cells in the post-layout design are colored based on their maximum occurring SHE above chip temperature. Although only 59 different standard cells are used in the design, a wide variety of SHE temperatures are observed. Besides specific transistor characteristics such as width or number of fins, the experienced SHE depends on input signal slews and connected load capacitance that differ for each standard cell in the circuit based on its position and pin connections. As a result, accurate estimation of SHE at the circuit-level is a challenging task. It is not implemented in commercial EDA tools. However, the functionality required to obtain SHE information (as depicted in Fig. 2) can be acquired with specifically adjusted standard cell libraries and conventional EDA tool-flows. An overview of our SHE flow is shown in Fig. 3.

The SHE flow outlined in Fig. 3 can be roughly separated into two sections. The upper part of the flow alters the standard cell characterization flow to include SHE temperatures in standard cell libraries. Before the characterization is carried out, additional SPICE instructions are injected to measure SHE temperatures for each operating condition and timing arc during the characterization process. Afterward, the obtained SHE temperatures are copied into the cell library, replacing the cell's delay information. With the SHE-included cell library, conventional STA can by employed to generate a standard delay format (SDF) file for any given circuit. Since the delays have been replaced with temperatures, the resultant SDF file no longer contains delays but the (maximum) SHE temperatures for each cell in the circuit [10]. The bottom part of the flow in Fig. 3 uses the individual cell SHEs to build a corner library that accurately reflects the impact of SHE on the target circuit. The idea is simple but hard to scale: Instead of characterizing each standard cell once for the entire circuit (as it is handled in conventional flows), we characterize each cell instance in the circuit under the impact of its corresponding SHE temperatures. This results in the library including thousands of cells, which is practically infeasible to use with conventional SPICE simulation. Therefore, we replace the conventional SPICE-based characterization with an ML-based approach that can generate a circuit-specific cell library including thousands of cells within seconds [9]. By using the resultant cell libraries in STA, we retrieve timing reports that yield accurate timing guardbands for the given circuit. In contrast to conventional worst-case estimations, this approach offers better circuit performance due to less pessimistic guardbands while still ensuring full reliability of the circuit during operation [11], [12].

Small yet sufficient guardbands are indispensable for efficient and reliable circuits. Traditional algorithms rely on these guarantees of correct computations. Circuits for approximate computing are designed intentionally with incorrect computation but are under the same reliability constraints as traditional circuits. To truly exploit the gains from technology scaling and emerging technologies, the reliability constrains have to be loosened and algorithms employed which function correctly despite unreliable circuits.

Brain-inspired hyperdimensional computing (HDC) is such an algorithm [13], [14]. Instead of computing with floating-

point weights like a neural network, HDC employs large vectors with thousands of components (hypervectors). Instead of fault-sensitive matrix multiplications, robust similarity computations of two hypervectors are performed. Despite an error rate of about 40 % on average, the inference accuracy with HDC drops only by 0.5 % [13]. This robustness against errors stems from the design of the hypervectors, which have independent and identically distributed (i.i.d.) components [14]. If HDC operations with some components of the hypervector are incorrect, due to the unreliable hardware, the other thousands of correctly performed operations prevent a failure because they are i.i.d. by design.

The HDC approach has been applied to a wide range of applications [15], from circuit reliability [16] and semiconductor manufacturing [17] to language and bio-signal classification [13]. HDC has also been employed to mimic confidential physics-based aging models [18]. Such detailed non-pessimistic models are not available to the circuit designer because the foundry calibrates them with their highly confidential technology parameters. Instead, designers have to rely on worst-case estimations for the whole chip. In the approach like [18], the foundry can train an HDC model with typical transistor stimuli as samples and the predicted transistor aging from their confidential model as the label. The HDC model learns to associate the gate voltage waveforms with the threshold voltage degradation ΔV_{th}. Because the model is based on hypervectors, it abstracts the sensitive information from the physics-based model. Thus, it can be made accessible to circuit designers, who, in turn, can employ a non-pessimistic aging model for close-to-the-edge guardband design.

III. Improving Architectural Reliability

A. Overview

With aggressive scaling of the semiconductor technology, high integration density aggravate the fault-rate of the device. Assessing the reliability of the safety-critical system against hardware faults accurately and quickly becomes a crucial design issue. The reliability of a target system can be evaluated in various ways, e.g., by either modeling the hardware faults or performing fault injection experiments. A key common challenge is the uncertainty associated with the occurrence of the actual faults, i.e., which architectural component becomes faulty and when during the application's execution. Simulating the architectural execution with faults injections can provide a more controlled setup for vulnerability analysis [19] and resilience of the architecture, but with slower simulations, the trials become much slower. Also, a huge number of injection trials are typically required for more accurate resilience analysis, which often makes simulations infeasible – at least for resource-constrained design environments or studies. Another approach is to inject faults within the application itself, which makes the analysis faster. But, faults simulated at higher levels of the computing stack can show much different behaviors and erroneous resiliency analysis than the real-world setup. This accuracy-efficiency trade-off remains a challenge in modeling faults at the architecture level.

Improving the reliability of a target system is also an important concern. This is usually resolved through redundancy, as defective parts of the system can be detected or corrected through their redundant counterparts. However, many application requirements cannot afford the immense overhead of full error protection techniques, especially embedded systems with strict resource-constraints. So, recent protection mechanisms sacrifice reliability at an extent with amortized overheads of performance and energy consumption. Selective replication techniques protect only the most vulnerable parts of the system to reduce overhead, but they rely on heuristics and probabilistic methods when determining the vulnerable parts of the system. Thus, they may not be broadly applied. Symptoms-based protection techniques detect errors by catching the flags that errors raise, but they tend to miss failure scenarios where there are no/mild fault-symptoms; thus they suffer from under-protection.

In this section, we discuss how ML-based approaches can be incorporated to help mitigate these challenges. ML models are effective and increasingly deployed for a wide range of tasks, including predictions and identifying patterns. With their approximation capabilities for arbitrary functions, they can help overcome limitations of heuristic models/measures for resilience. Further, as even simple models may accurately do predictions, they can reduce overheads of performing time-consuming fault injections or error mitigations. For instance, light-weight models for robust and secure execution can be part of the architecture itself, and can be repeatedly invoked at every few cycles as compared to intensive/redundant error checking in the hardware or software. ML-based approaches can also help to distinguish between the reliable and vulnerable regions of the large-scale system and selectively safe-guarding it.

In summary, this section discusses the use of ML techniques to achieve the following:

- Resolving the challenges in modeling and analyzing faults at the architecture level
- Developing accurate and efficient reliability-enhancing techniques

B. Modeling Reliability using ML

Techniques for statistical fault analysis mitigate the infeasible task of collecting and analyzing the results for all possible fault injections. However, the number of trials for fault injections still needs to be large enough so that the analysis is statistically significant. Further, the sheer amount of data makes it difficult to do analysis and acquire insights about how the errors propagate. In this section, we discuss how ML models can help mitigate such challenges.

1) Accelerating Fault Injection Process: Architecture's vulnerability to the hardware-faults can be modeled by injecting the faults into the flip-flops of the circuit. However, fault injections into each flip-flop are infeasible. In such scenarios, simple ML models like k-nearest neighbor or support vectors can be trained that predicts whether the flip-flop is vulnerable to the fault or not. The training data for these models can account for relevant features, e.g., fan-in/fan-out factors and information about connections and proximity of flip-flop's

inputs/outputs [20]. Recent works such as [20] show that these ML models can predict vulnerabilities with similar accuracy while using about only 20% of the data for the training. Thus, the overall fault injection process can be accelerated by a considerable factor.

In large-scale environments, even one injection trial may become too slow. One approach is to model errors at a small scale and then using it with ML-based techniques to predict the error behavior at the large scale. For instance, [21] showed that the fault behaviors of large-scale applications like DOE applications running on 4096 cores can be modeled with 90% accuracy while using the data from the small-scale execution on a single core. As compared to simpler ML models like multi-layer perceptrons (MLPs), naive bayes, or support vector machines, ML models like AdaBoost or stochastic gradient boosting can be more consistently accurate, as they continuously learn from mispredicted samples and adapt their weights [21].

2) Resiliency Analysis: Extracting meaningful information from the collected large data requires an immense amount of effort. This can be overcome by using ML models to find patterns and predict potential fault scenarios. This is because, ML models can filter irrelevant features of execution traces and draw correlations between the fault-injection outcomes and platform's architectural characteristics. For instance, [22] showed the effectiveness of the gradient boosting decision tree for finding error patterns in a large trace data collected from a large-scale HPC system during 6 months. The decision-tree-based model also predicted future scenarios of errors in GPU executions. Likewise, [23] showed using various supervised and unsupervised learning techniques to find patterns in a dataset containing over 1.2 million fault injection trials.

ML-based techniques for fault prediction or silent data corruption (SDC) often rely on handcrafted features that may not entirely capture the valuable information for predicting the errors. This can be automated by predicting the proneness of instructions to the errors through ML models. For example, the graph attention network in [24] predicts SDC-prone instructions. It considers a program as a heterogeneous graph in which a node is an instruction and different types of edges represent the relationship between the instructions. This is because, the effects of fault propagation vary depending on the relationships among different instructions. Its graph neural network learns hidden structural features by aggregating the neighboring nodes' features (opcode and the destination operands), and then reasons about contributions of nodes to fault propagation through a self-attention mechanism. Then, it predicts the probability of the outcome (SDC, crash, hang, benign fault) through a softmax function. For generalization, [24] extends the graph neural network to an inductive model, so that the trained model can be applied to unknown programs without retraining and additional fault injection experiments for the target programs.

C. Improving Reliability using ML

1) Selective Replication: The most vulnerable parts of the system can be protected by selective replication. To do so, the reliability of each part of the system must be examined. This is usually achieved through extensive fault injection experiments or heuristics-based analysis. In this section, we discuss how ML-based techniques can greatly enhance the accuracy and speed of this step.

Software-based approaches for error-resilient computing typically replicates instructions for detecting and protecting errors [25], [26]. ML-based approaches can help identify the vulnerable instructions that need to be replicated instead of replicating all instructions. For example, IPAS [27] extracted features from each instruction and performed random fault injection trials to find vulnerable and non-vulnerable instructions. The fault injection results and instruction features were fed to train a support vector machine (SVM). By replicating only the instructions classified as vulnerable by the SVM, IPAS achieved as much as 47% less slowdown compared to the baseline selective replication technique, while maintaining similar coverage.

Similar technique can be applied at architectural level. Identifying the faults that are critical to functionality when processing applications on an architecture can help limit the redundancy that need to be introduced in the architecture. For instance, for processing deep neural networks (DNNs) on memristor crossbars, [28] trained a small neural network to predict the criticality of a fault with 99% accuracy. By protecting only the critical faults, it reduced the redundancy required in crossbars for fault tolerance by 93%.

2) Symptom-based Detection: Symptom-based detection techniques assume that the errors that eventually lead to failure would show specific symptoms during the execution of the system. However, these symptoms were previously determined through heuristics and previous symptom-based detection suffered low error-coverage rates [29]. As ML models can effectively detect patterns, recent studies suggest using ML-based techniques to detect even the mildest of symptoms of failure.

Small models like MLPs are often used to identify signs of errors in executions. For example, [30] used a neural network with two hidden layers for detecting anomalies in the intermediate outputs, when executing DNNs. It detected misclassification errors with 99% recall and 97% precision while requiring only 2.67% computation overhead compared to executing DNNs with no protection. The reason behind usage of such simpler models for resiliency is their effectiveness of capturing error propagation without introducing much computational overhead. Plus, such models can be efficiently processed on existing processors. Further, in case such ML models for resiliency are larger, they can still be compressed to avoid computational/storage overheads while achieving resilience goals. This is because recent compression techniques for ML models make them significantly compact so that their computations and storage can be reduced by orders of magnitude with achieving similar accuracy for predictions [31].

In addition to detecting the symptoms of errors during the execution of the system, ML-based techniques can be used to detect perturbations in the inputs to the system. For example, WarningNet [32] is a small neural network that could be executed in parallel with a mission-critical task to detect noise

or environmental conditions in the input that could lead to task failures. Due to its simplicity, WarningNet consumes only about 1/20-th of the time to detect potential failures. The reliability of the overall system can be improved by the early warnings provided with on-demand pre-processing of the input.

IV. Improving Application Reliability through OS

Due to technology scaling, the safety and reliability issues have increased tremendously, which often increase aging, lead to permanent faults, and cause unreliable execution of applications. Transient faults lead to soft errors, which may result in incorrect execution or crashing of applications. In general, *soft error rate (SER)* can be expressed as the number of failures in a given time, and it may degrade the functional or timing reliability of applications [1], [33]. Failure during an application execution in safety-critical embedded systems like avionics may cause catastrophic consequences, which is not admissible. Therefore, managing reliability of applications under all circumstances of stress and environmental changes is crucial in these systems during run-time [34]–[36].

In order to improve the reliability in these embedded systems, several system-level design- and run-time techniques have been exploited. However, most of the existing approaches are not adaptive to variations in system conditions and workloads during run-time, or they need prior knowledge of the system and applications that introduce large profiling overheads [4], [37]. ML techniques are recently being employed for dynamic reliability improvement, as they can adapt to variations in workloads and system conditions. They can learn from past events and make better decisions to improve the system's performance while sustaining the system's operation.

ML techniques can be classified into three categories: *supervised learning*, *unsupervised learning*, and *reinforcement learning*. In *supervised learning*, a function is extrapolated from given inputs and outputs at design-time to learn the relation between them. It is then used to determine action at run-time based on the learned data in previous experiences. Linear Regression and Neural Network are the most common supervised-learning methods, used for reliability improvement [2]. Contrarily, *unsupervised learning* extrapolates a function from only the input data by working on its own to learn and determine the output at run-time. It often achieves lesser accuracy while being computationally intensive. In commonly used *reinforcement learning*, learning is through trial-and-error interaction with an environment. The interaction between the environment and agent is modeled using finite states, a set of actions, and a reward function [4], [38]. Pagani *et al.* [4] presented a comprehensive study of learning techniques. In general, depending on the problem, parameters, and inputs, at least one of these techniques is used for system optimization.

ML-based reliability improvement techniques commonly consider three OS-level knobs or a combination thereof:

- *Task-to-core (re-)allocation* used in multi/many-core processors, including homogeneous and heterogeneous cores, is an effective method for system optimization. Survey [34] discussed various mapping methodologies depending on different optimization goals. Such an approach provides flexibility to control the peak temperature and thermal-cycling based on the temperature at run-time, and consequently, the device failure is decelerated [39], [40].
- *Dynamic voltage and frequency scaling (DVFS)* is an effective and commonly used technique that allows the cores to scale their voltage and/or frequency *V-f* levels dynamically; it can be applied to cores individually, in clusters, or globally, based on the target architecture. Although DVFS is an optimization technique for minimizing power, preventing temperature gradients and hot-spots, and consequently maximizing the lifetime reliability, it negatively affects the functional reliability because of the increase in transient fault rate and tasks' execution time [1], [41]. Therefore, the DVFS-based optimizations need to account for the trade-off between the lifetime and functional reliability.
- *Dynamic power management (DPM)* can change the power states of the system's cores into active, idle, sleep, or off modes. While this technique is typically used for improving energy efficiency, it can also help manage the thermal and reliability issues, especially by tuning the state of cores in multi/many-core processors [42].

If the control knobs are not configured effectively, or the appropriate learning technique is not used, it could violate functional or timing reliability or accelerate aging which degrades the lifetime for reliable computing. Knowing about the target applications and requirements for the system can help to select most effectual configuration for error-resilient computing on different platforms and under different working environments.

Recent approaches apply ML-based techniques for improving dynamic reliability in single/multi/many-core processors. In general, they configure the design knobs to improve the system reliability, in addition to other optimization goals and constraints, such as average/peak power, temperature, or performance. Rest of this section summarizes such approaches that aim to improve reliability through learning.

A. Timing and Functional Reliability Improvement

ML-based approaches improve timing and functional reliability in the following ways:

1) Soft Error Mitigation: System reliability is negatively affected due to increased fault rate while using the DVFS optimization knob and lowering the *V-f* levels to optimize energy consumption. Therefore, some approaches like [33], [43] target learning-based dynamic reliability management, while minimizing the energy consumption/maximizing the lifetime reliability at run-time under SER, performance, temperature, and power constraints. A neural network can be trained for quick and accurate SER estimation [43]. Such approaches target soft errors and MTTF separately, and they suffer a drawback when real-time applications are executing. Applying the DVFS technique causes long execution of applications and a negative impact on timing reliability.

2) Application Reliability Improvement: To optimize the functional reliability of applications, the impact of soft errors, i.e., transient faults, must be investigated during the execution of tasks, e.g., as considered in [1], [44]. [1] has investigated

the dynamic cross-layer (component and system layers) SER model based on a neural network, which is trained by using data obtained through SPICE simulation. The availability of the system (i.e., MTTF, affected by both soft and hard errors) is then optimized during run-time for safety-critical real-time systems by applying the DVFS-based learning method. The application and thermal reliability can be improved by using the reinforcement learning on the DVFS-enabled multi-core platform [44]. Such ML-based manager is shown to improve the functional reliability with no timing constraints for applications, which currently makes it infeasible for safety-critical tasks.

3) Mean Workload to Failure Optimization: In order to optimize the *mean workload to failure (MWTF)* at run-time, some approaches like [2] consider the optimal task-to-core mapping in heterogeneous multi-core processors. For example, [2] showed that by maximizing the MWTF, more tasks can be executed successfully before the system fails; this could be estimated by considering the Architectural-Vulnerability-Factor, the task's execution time, and the raw SER. It used a neural network to estimate vulnerability factors of heterogeneous cores for obtaining an efficient task mapping and balancing between performance and vulnerability, while maximizing the MWTF.

4) Replicas Management: To improve functional reliability, fault-tolerance techniques such as replication are been used that could guarantee the correct execution of real-time tasks. ML-based approaches can be employed to determine the status of architecture, whether it is faulty or non-faulty, modify the fault-tolerance attributes, and change the number of task replicas in response to environmental changes e.g. in [45].

B. Lifetime Reliability Improvement

1) MTTF Maximization: In determining how reliable a system is over the long term (called lifetime reliability), and how long an application can execute safely, some metrics such as MTTF can be used, depending on whether a fault can be repaired. To model the system-level lifetime reliability and estimate the MTTF, the designers use one of the device-level reliability models, such as electro-migration (EM), temperature-dependent dielectric breakdown (TDDB), thermal-cycling (TC), negative bias temperature instability (NBTI), and hot carrier injection (HCI). A comprehensive study of device-level reliability models and their equations is presented in [46]. Techniques either improve the system-level lifetime reliability simultaneously with other design parameters/metrics, or they optimize other system parameters/metrics under the lifetime reliability constraint. We focus on the recent works in the first category. In order to optimize MTTF, several schemes based on device-level reliability models have been proposed, including:

- TDDB lifetime reliability improvement through ML-based enhancement of the system's availability, which is affected by TDDB and soft errors [1]. The enhancement is based on the reinforcement learning and uses DVFS knob under the timing constraints and system utilization bound in single-core safety-critical systems
- EM lifetime reliability improvement [33], [44]. The improvement is based on the reinforcement learning and employing the DVFS and DPM knobs to control the active cores under peak power, temperature, and performance constraints.
- TC lifetime reliability improvement through ML-based thermal management while preserving the performance constraint [39], [40]. The improvement is based on the reinforcement learning and using thread allocation and DVFS knobs to address thermal-cycling, peak or average temperature.
- NBTI lifetime reliability improvement by proposing an ML-based process variation and aging-aware approach [47]. The improvement is based on the reinforcement learning and using task-to-core mapping and DVFS knobs.
- BTI stress estimation and mitigation at system level through learning [48]. With the trained model, the stress can be mitigated by mapping the workload and adjusting core frequency appropriately.

According to the discussion, the DVFS is a useful technique to optimize MTTF and lifetime reliability. However, the timing and functional reliability are degraded with decreasing the *V-f* levels, which needs to be considered. From the perspective of learning methods, a lightweight ML technique should be employed that could make accurate predictions and effective decisions, even at scale, e.g., with an increased number of cores, inputs, or objectives. Thus, using some learning methods may not be efficient due to the memory and timing overheads.

2) Thermal Management: Many studies have focused on reducing the thermal hot-spots and high-temperature gradients due to its affect on MTTF and lifetime reliability. Recent research works have focused on maximum temperature reduction through different approaches such as 1) learning-based task-to-core allocation [49], 2) supervised-learning-based thermal management through monitoring temperatures of the cores and using the DVFS and DPM knobs [50].

Since thermal stress is one of the major causes of declining lifetime reliability, both spatial and temporal thermal gradients are suggested to be controlled in multi/many-core platforms, which are not studied in most recent works. DVFS and task re-allocation are the two most common techniques used for thermal management. However, most of the embedded systems are real-time; the timing overheads of changing the *V-f* levels and migration are the major concerns during run-time, which have not been studied attentively in ML-based approaches. Besides, the timing overhead of predicting/measuring the temperature, which may be used in learning techniques, is also essential to consider. In addition, ML techniques can be memory-intensive and computationally expensive. It makes some of these techniques incompatible with real-time systems and resource-constrained embedded systems. While a simpler ML model could be usable, it may not necessarily provide the desired accuracy when making predictions. Therefore, choosing an appropriate ML technique and accounting for its run-time overheads for learning and prediction play a key role in controlling a safe mission.

V. Reliability Analysis on a Fault-Tolerant Timing-Guaranteed System

In logic circuits, errors such as timing violations in critical paths could occur at registers of every pipeline stage due to

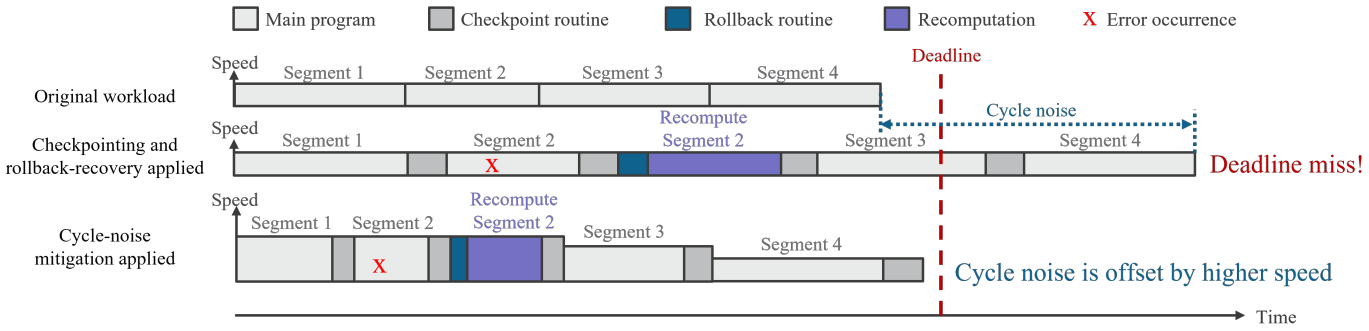

Fig. 4. Schematic of the collaboration between checkpointing and rollback-recovery, and cycle-noise mitigation.

variability in process, voltage, temperature, and aging. To ensure functional correctness, these errors need to be corrected by fault-tolerant mechanisms such as checkpointing and rollback-recovery [51]. However, these mechanisms induce execution time overhead by generating "cycle noise" [52], which is the variability in the number of clock cycles to run an application. This is particularly detrimental to time-critical applications such as multimedia and autopilot, which are required to finish before their deadlines. To ensure timing guarantees of these systems, the cycle noise needs to be mitigated. This could be achieved by real-time scheduling which predicts the future workload and switches processor speed accordingly at run time [52].

For time-critical applications, two reliability requirements should be simultaneously fulfilled: functional correctness by error correction, and timing guarantees by preventing deadline misses. Fig. 4 shows how a checkpointing and rollback-recovery mechanism could collaborate with a cycle-noise mitigation mechanism to meet the dual reliability requirements. Without cycle-noise mitigation, the errors could be fixed by rollback and re-computation of the erroneous parts. However, the generated cycle noise might cause deadline misses. With cycle-noise mitigation, the processor speed are raised in advance in consideration of potential rollback events, so the deadline misses are prevented. Both mechanisms could be implemented with reasonable overhead of time, energy and area [51], [53]. Moreover, cycle-noise mitigation system can be optimized by learning-based approaches to improve its prediction accuracy of execution time.

We analyzed the interplay of both mechanisms with a model we developed. The details are elaborated in the following parts.

A. Register-level error model

Prior works addressed real-time scheduling in the presence of errors [54], [55]. However, most of them assume only a single error or a bounded number of errors, and they limit the error occurrences to only the main program, but not the re-computation time. These limitations fall short in reflecting the statistics of error occurrences.

To be realistic, the error model in our analysis does not limit the number of errors or the time of errors. In this model, a cycle is erroneous if any register of a pipeline stage contains a wrong value. The probability that a cycle is erroneous is static over time, which represents the effect of variability in logic circuits at the running time of the application. The probability that there is no error during any time interval is as follows:

$$Pr(N_e = 0) = (1-p)^{n_c} \tag{1}$$

where N_e is the number of erroneous cycles, n_c is the total number of cycles in the interval, and p is the probability of a cycle to be erroneous, respectively.

B. Checkpointing and rollback-recovery system

A timing model of a checkpointing and rollback-recovery system is created by abstracting the HW/SW reliability mitigation approach [51]. In this model, each application is segmented into atomic units. A checkpoint routine of 100 cycles is performed in the end of each segment, which is similar to [51]. If any error occurs during the running time of this segment, a rollback routine of 48 cycles [51] is inserted, and then the segment is recomputed. Each re-computation must be followed by another checkpoint routine, and possibly another instance of rollback and re-computation if there is still error during the previous re-computation. Therefore, there is no bound of the number of re-computation. The number of rollbacks for each segment follows the geometric distribution, derived with (1):

$$Pr(N_{rb} = n_{rb}) = (1-(1-p)^{n_c})^{n_{rb}}\,(1-p)^{n_c} \tag{2}$$

where N_{rb} is the number of rollbacks.

C. Cycle-noise mitigation system

Multi-timescale performance variability mitigation approach [53] is adopted for cycle-noise mitigation. Different scheduling algorithms can be applied in this approach, depending on the budget (execution time and processor speed) allocated for each segment of the application. In general, conservative algorithms supply segments with larger budgets, which could absorb cycle noise. Therefore, conservative algorithms are better at meeting deadlines in the presence of errors. However, the high processor speed used by conservative algorithms consumes more energy, which is unnecessary if errors are rare.

Four algorithms are selected in our analysis, ranging from aggressive ones to conservative ones:

- **DS**: dynamic-scenario based ***(most aggressive)***
- **DS 1.5x**: dynamic-scenario based, budgets scaled by 1.5×
- **DS 2x**: dynamic-scenario based, budgets scaled by 2×
- **WCET**: worst-case execution time ***(most conservative)***

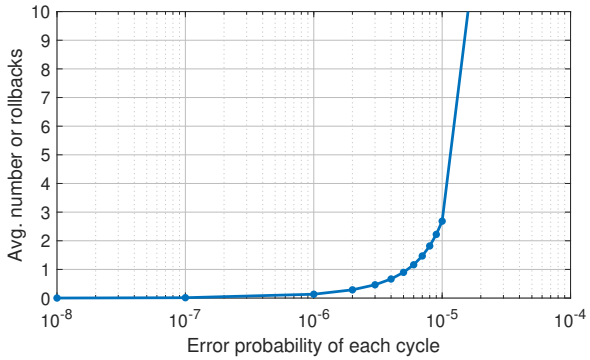

Fig. 5. Average number of rollbacks each segment generated by the checkpointing and rollback-recovery system, under different error probability conditions.

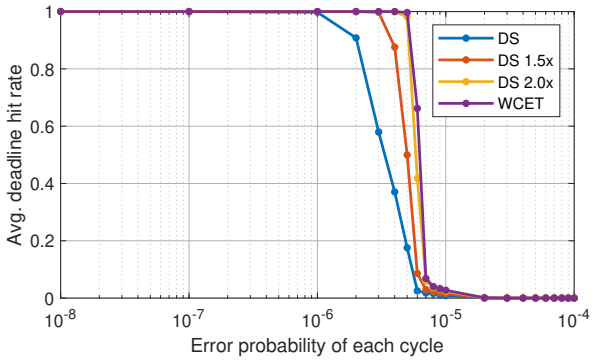

Fig. 6. Average deadline hit rate under different error probability conditions, using different scheduling algorithms for cycle-noise mitigation.

Dynamic-scenario based approach [53] is the most aggressive algorithm, which can derive a tight budget at run time. On the other hand, worst-case execution time [56] is the most conservative algorithm. In addition, two variants of the dynamic-scenario based algorithms are created, in which the budgets for all segments are enlarged by $1.5\times$ and $2\times$ respectively.

D. Simulation results and discussion

Our analysis used the lower sub-band quantization block of the ADPCM-encoding application in TACLeBench [57] as the workload. It was benchmarked on the RTL model of the Ariane processor core [58] and then segmented into segments of 40k-270k cycles. Number of rollbacks for each segment were randomly generated by the probability distribution of (2). We performed simulations across different levels of error probability. For each level, we performed Monte Carlo simulations of 100 runs and calculated the average results.

Fig. 5 shows the average number of rollbacks each segment generated by the checkpointing and rollback-recovery system. In our demonstrated system, the number of rollbacks increases rapidly beyond the error probability of 10^{-6}. When the error probability exceeds 10^{-5}, the number of rollbacks increases to more than 10 rollbacks per segment, which is formidable to deal with. The results indicate that an "error rate wall" exists around 10^{-6} to 10^{-5} error probability, under which the reliability requirements can be fulfilled. The position of the wall is strongly dependent on system parameters, such as the processor speed, the granularity of checkpointing, etc. Therefore, it is possible to improve the error rate limit (i.e. moving the wall forward) by optimizing the system. For example, it is shown that execution time overhead can be minimized by optimizing the number of checkpoints [51].

Fig. 6 shows that the deadline hit rate, which represents the capability of ensuring timing guarantees, is also highly sensitive to the error probability. In a small window of 10^{-6} to 10^{-5} error probability, the deadline hit rates drop from almost one to almost zero. For the scheduling algorithms used in cycle-noise mitigation, the results show that within this window, conservative algorithms bring higher deadline hit rates. However, when the error probability further increases, all deadline hit rates converge to zero regardless of the algorithms. These results are in agreement with the previous observation of the wall around 10^{-6} to 10^{-5} error probability. Within the error rate wall, the execution time overhead could be offset by higher processor speed, so deadline misses could still be prevented by scheduling algorithms. However, beyond the error rate wall, the rollbacks are so frequent that even the highest processor speed cannot save the deadlines from being missed. Thus, the error rate wall defines the inherent system limit of the conditions that both reliability requirements can be fulfilled. Determining how system parameters (e.g. processor speed) affect the error rate wall remains an important future work.

VI. Upcoming Trends and Open Challenges

In this section, we discuss the open challenges and limitations for future academic and industrial works.

A. Run-time Cross-Layer Reliability Improvement

Recent approaches consider the cross-layer reliability, investigating the reliability, faults, and soft error mitigation of different layers [3], [35]. Due to the system's unexpected behavior, the faults in each layer may propagate and consequently cause an error manifestation, which impacts the other layers' reliability. Available solutions for improving cross-layer reliability can lead to an explosion in the design complexity because a number of configurations could be effective for reliable computing at each layer. Thus, developing dynamic learning mechanisms for improving cross-layer reliability is an important open challenge.

B. Reliability Improvement in Mixed-Criticality Systems

Mixed-criticality systems are widely used in various industrial applications, e.g., medical devices and avionics, where tasks are classified into multiple criticality levels in terms of real-time and reliability/safety requirements for maintaining the applications' predictability under different (often unseen) circumstances. These systems have different operational modes at run-time, and hence, the reliability needs of various applications in each criticality level must be guaranteed in these circumstances to prevent damages [59], [60]. However, due to environmental changes, reliability requirements may be affected, which cause catastrophic consequences. To further ensure the

timing and lifetime reliability, ML techniques with low run-time timing overheads need to be applied for identifying the application trend and optimizing the system reliability.

C. Selection of ML Models for Modeling/Improving Resilience

ML models for modeling architectural vulnerability or fault prevention are typically supervised and require abundant data for the training in order to be effective. Future approaches could investigate ML models that require small amount of samples or are unsupervised and adapt as new faults get identified. In addition, approaches could characterize the effectiveness of applying linear and non-linear models in modeling resilience and protecting against faults so that system designers can easily identify the ML models for their application-platform configuration. Further, as features accounted by ML models grow, dimensionality reduction techniques should be applied for meaningful resiliency analysis and mitigation.

D. Applying ML Models to Different Design Phases and Applications

Current ML-based approaches, especially for architectural-level reliability modeling, primarily focus on accelerating fault injection experiments. However, collecting the training data for these ML models remains to be accelerated. So, generative models or ML-based techniques to accelerate data collection can be useful. Moreover, recent ML-based methods for improving architectural reliability has focused on specific domains such as deep learning [61], presumably because of the ease of collecting critical faults or detecting anomalous behavior. Given the potential of ML-based approaches for fault tolerance, their broad applicability to other vulnerable general-purpose applications can be explored.

VII. Summary

The paper provided insights into how machine learning techniques can be employed to improve reliability in different abstraction layers from the transistor layer to the application layer. We also studied the reliability and timing costs of correcting register-level errors. Lastly, the paper discussed trends and open challenges for improving error-resiliency. We anticipate ML models to improve cross-layer reliability and provide low-cost fault tolerance with distributed fault mitigation activity across the layers.

Acknowledgement

This research was supported in part by 1) Advantest as part of the Graduate School "Intelligent Methods for Test and Reliability" (GS-IMTR) at the University of Stuttgart; 2) funding from National Science Foundation Grants No. CPS 1646235, CCF 1723476 – the NSF/Intel joint research center for Computer Assisted Programming for Heterogeneous Architectures (CAPA) at Arizona State University; 3) National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2022-00165225) at Yonsei University; 4) German Research Foundation (DFG) within the Cluster of Excellence Center for Advancing Electronics Dresden (CFAED) at the Technische Universität Dresden.

References

[1] L. Li, J. Zhou, T. Wei, M. Chen, and X. S. Hu, "Learning-Based Modeling and Optimization for Real-time System Availability," *IEEE Trans. on Computers (TC)*, vol. 70, no. 4, pp. 581–594, 2021.

[2] R. B. Tonetto, M. d. A. Hiago, G. L. Nazar, and A. C. S. Beck, "A machine learning approach for reliability-aware application mapping for heterogeneous multicores," in *57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[3] S. S. Sahoo, A. Kumar, M. Decky, S. C. B. Wong, G. V. Merrett, Y. Zhao, J. Wang, X. Wang, and A. K. Singh, "Emergent Design Challenges for Embedded Systems and Paths Forward: Mixed-Criticality, Energy, Reliability and Security Perspectives," in *Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES)*, 2021, p. 1–10.

[4] S. Pagani, P. D. S. Manoj, A. Jantsch, and J. Henkel, "Machine Learning for Power, Energy, and Thermal Management on Multicore Processors: A Survey," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 1, pp. 101–116, 2020.

[5] H. Amrouch and J. Henkel, "Lucid infrared thermography of thermally-constrained processors," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2015, pp. 347–352.

[6] G. Paim, L. M. G. Rocha, H. Amrouch, E. A. C. da Costa, S. Bampi, and J. Henkel, "A Cross-Layer Gate-Level-to-Application Co-Simulation for Design Space Exploration of Approximate Circuits in HEVC Video Encoders," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 30, no. 10, pp. 3814–3828, 2020.

[7] H. Amrouch, S. Salamin, G. Pahwa, A. D. Gaidhane, J. Henkel, and Y. S. Chauhan, "Unveiling the impact of ir-drop on performance gain in ncfet-based processors," *IEEE Trans. on Electron Devices*, vol. 66, no. 7, pp. 3215–3223, 2019.

[8] M. Rapp, S. Salamin, H. Amrouch, G. Pahwa, Y. Chauhan, and J. Henkel, "Performance, Power and Cooling Trade-Offs with NCFET-Based Many-Cores," in *Proc. 56th Ann. Design Automation Conference (DAC)*, 2019.

[9] F. Klemme and H. Amrouch, "Machine learning for on-the-fly reliability-aware cell library characterization," *IEEE Trans. on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 68, no. 6, pp. 2569–2579, 2021.

[10] F. Klemme, S. Salamin, and H. Amrouch, "Upheaving Self-Heating Effects from Transistor to Circuit Level using Conventional EDA Tool Flows," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[11] F. Klemme and H. Amrouch, "Machine Learning for Circuit Aging Estimation under Workload Dependency," in *IEEE International Test Conference (ITC)*, 2021, pp. 37–46.

[12] F. Klemme *et al.*, "Scalable Machine Learning to Estimate the Impact of Aging on Circuits Under Workload Dependency," *IEEE Trans. on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 69, no. 5, pp. 2142–2155, 2022.

[13] S. Thomann, H. L. G. Nguyen, P. R. Genssler, and H. Amrouch, "All-in-Memory Brain-Inspired Computing Using FeFET Synapses," *Frontiers in Electronics*, vol. 3, 2022.

[14] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation*, vol. 1, no. 2, pp. 139–159, 2009.

[15] L. Ge and K. K. Parhi, "Classification Using Hyperdimensional Computing: A Review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[16] P. R. Genssler and H. Amrouch, "Brain-inspired computing for circuit reliability characterization," *IEEE Transactions on Computers*, vol. 71, no. 12, pp. 3336–3348, 2022.

[17] ——, "Brain-Inspired Computing for Wafer Map Defect Pattern Classification," in *IEEE International Test Conference (ITC)*, 2021, pp. 123–132.

[18] P. R. Genssler, H. E. Barkam, K. Pandaram, M. Imani, and H. Amrouch, "Modeling and predicting transistor aging under workload dependency using machine learning," *arXiv preprint arXiv:2207.04134*, 2022.

[19] K. Tanikella, Y. Koy, R. Jeyapaul, K. Lee, and A. Shrivastava, "gemV: A validated toolset for the early exploration of system reliability," in *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2016, pp. 159–163.

[20] T. Lange, A. Balakrishnan, M. Glorieux, D. Alexandrescu, and L. Sterpone, "On the estimation of complex circuits functional failure rate by machine learning techniques," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks – Supplemental Volume (DSN-S)*, 2019, pp. 35–41.

[21] G. Kestor, I. B. Peng, R. Gioiosa, and S. Krishnamoorthy, "Understanding scale-dependent soft-error behavior of scientific applications," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2018, pp. 482–491.

[22] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari, "Machine learning models for gpu error prediction in a large scale hpc system," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018.

[23] F. R. da Rosa, R. Garibotti, L. Ost, and R. Reis, "Using machine learning techniques to evaluate multicore soft error reliability," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 6, 2019.

[24] J. Ma, Z. Duan, and L. Tang, "Deep Soft Error Propagation Modeling Using Graph Attention Network," *Journal of Electronic Testing*, 2022.

[25] A. Shrivastava and M. Didehban, "Software approaches for in-time resilience," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–4.

[26] M. Didehban, A. Shrivastava, and S. R. D. Lokam, "NEMESIS: A software approach for computing in presence of soft errors," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 297–304.

[27] I. Laguna, M. Schulz, D. F. Richards, J. Calhoun, and L. Olson, "Ipas: Intelligent protection against silent output corruption in scientific applications," in *2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2016, pp. 227–238.

[28] C.-Y. Chen and K. Chakrabarty, "Efficient identification of critical faults in memristor crossbars for deep neural networks," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1074–1077.

[29] H. So, M. Didehban, Y. Ko, R. Jeyapaul, J. Kim, Y. Kim, K. Lee, and A. Shrivastava, "Revisiting symptom-based fault tolerant techniques against soft errors," *Electronics*, vol. 10, no. 23, p. 3028, 2021.

[30] C. Schorn, A. Guntoro, and G. Ascheid, "Efficient on-line error detection and mitigation for deep neural network accelerators," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2018, pp. 205–219.

[31] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, "Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights," *Proceedings of the IEEE*, vol. 109, no. 10, pp. 1706–1752, 2021.

[32] M. Lee, B. Mudassar, T. Na, and S. Mukhopadhyay, "Warningnet: A deep learning platform for early warning of task failures under input perturbation for reliable autonomous platforms," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[33] T. Kim, Z. Sun, H. Chen, H. Wang, and S. X.-D. Tan, "Energy and Lifetime Optimizations for Dark Silicon Manycore Microprocessor Considering Both Hard and Soft Errors," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 9, pp. 2561–2574, 2017.

[34] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–10.

[35] S. S. Sahoo, B. Ranjbar, and A. Kumar, "Reliability-aware resource management in multi-/many-core systems: A perspective paper," *Journal of Low Power Electronics and Applications*, vol. 11, no. 1, p. 7, 2021.

[36] C. K. Pang, A. Kumar, C. H. Goh, and C. V. Le, "Nano-satellite swarm for sar applications: design and robust scheduling," *IEEE Trans. on Aerospace and Electronic Systems*, vol. 51, no. 2, pp. 853–865, 2015.

[37] A. Iranfar, F. Terraneo, G. Csordas, M. Zapater Sancho, W. Fornaciari, and D. Atienza Alonso, "Dynamic Thermal Management with Proactive Fan Speed Control Through Reinforcement Learning," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 418–423.

[38] B. Ranjbar, H. Alikhani, B. Safaei, A. Ejlali, and A. Kumar, "Learning-Oriented QoS- and Drop-Aware Task Scheduling for Mixed-Criticality Systems," *Computers*, vol. 11, no. 7, 2022.

[39] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems," in *Proc. 51st Ann. Design Automation Conference (DAC)*, 2014, p. 1–6.

[40] A. Das, B. M. Al-Hashimi, and G. V. Merrett, "Adaptive and Hierarchical Runtime Manager for Energy-Aware Thermal Management of Embedded Systems," *ACM Trans. on Embedded Computing Systems (TECS)*, vol. 15, no. 2, 2016.

[41] B. Ranjbar, A. Hosseinghorban, M. Salehi, A. Ejlali, and A. Kumar, "Toward the Design of Fault-Tolerance-Aware and Peak-Power-Aware Multicore Mixed-Criticality Systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 5, pp. 1509–1522, 2022.

[42] Z. Yang, C. Serafy, T. Lu, and A. Srivastava, "Phase-driven learning-based dynamic reliability management for multi-core processors," in *54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[43] Z. Sun, H. Zhou, and S. X.-D. Tan, "Dynamic Reliability Management for Multi-Core Processor Based on Deep Reinforcement Learning," in *Int. Conf. on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2019, pp. 217–220.

[44] S. M. P. Dinakarrao, A. Joseph, A. Haridass, M. Shafique, J. Henkel, and H. Homayoun, "Application and Thermal-Reliability-Aware Reinforcement Learning Based Multi-Core Power Management," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 4, pp. 1–19, 2019.

[45] A. Ballesteros, J. Proenza, and P. Palmer, "Towards a dynamic task allocation scheme for highly-reliable adaptive distributed embedded systems," in *22nd IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–4.

[46] A. K. Das, A. Kumar, B. Veeravalli, and F. Catthoor, *Reliable and Energy Efficient Streaming Multiprocessor Systems*. Springer, 2018.

[47] V. Rathore, V. Chaturvedi, A. K. Singh, T. Srikanthan, and M. Shafique, "Life Guard: A Reinforcement Learning-Based Task Mapping Strategy for Performance-Centric Aging Management," in *56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[48] H. M. Abbas, B. Halak, and M. Zwolinski, "Learning-based BTI stress estimation and mitigation in multi-core processor systems," *Microprocessors and Microsystems*, vol. 81, p. 103713, 2021.

[49] S. J. Lu, R. Tessier, and W. Burleson, "Reinforcement Learning for Thermal-Aware Many-Core Task Allocation," in *Proc. 25th Edition on Great Lakes Symposium on VLSI (GLSVLSI)*, 2015, p. 379–384.

[50] S. Dey, A. K. Singh, X. Wang, and K. McDonald-Maier, "User Interaction Aware Reinforcement Learning for Power and Thermal Efficiency of CPU-GPU Mobile MPSoCs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 1728–1733.

[51] M. M. Sabry, D. Atienza, and F. Catthoor, "OCEAN: An Optimized HW/SW Reliability Mitigation Approach for Scratchpad Memories in Real-Time SoCs," *ACM Trans. Embed. Comput. Syst. (TECS)*, vol. 13, no. 4s, pp. 1–26, 2014.

[52] M. Noltsis, D. Rodopoulos, N. Zompakis, F. Catthoor, and D. Soudris, "Runtime Slack Creation for Processor Performance Variability Using System Scenarios," *ACM Trans. Des. Autom. Electron. Syst. (TODAES)*, vol. 23, no. 2, pp. 1–23, 2017.

[53] J.-Y. Lin, P. Weckx, S. Mishra, A. Spessot, and F. Catthoor, "Multi-timescale Mitigation for Performance Variability Improvement in Time-Critical Systems," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 11, pp. 1757–1769, 2022.

[54] H. Lee, H. Shin, and S.-L. Min, "Worst case timing requirement of real-time tasks with time redundancy," in *Proc. 6th Int. Conf. on Real-Time Computing Systems and Applications (RTCSA) (Cat. No.PR00306)*, 1999, pp. 410–414.

[55] R. Melhem, D. Mosse, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Trans. on Computers (TC)*, vol. 53, no. 2, pp. 217–231, 2004.

[56] P. Pillai and K. G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, p. 89–102, 2001.

[57] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Wägemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *16th Int. Workshop on Worst-Case Execution Time Analysis (WCET)*, 2016.

[58] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.

[59] B. Ranjbar, B. Safaei, A. Ejlali, and A. Kumar, "FANTOM: Fault Tolerant Task-Drop Aware Scheduling for Mixed-Criticality Systems," *IEEE Access*, vol. 8, pp. 187 232–187 248, 2020.

[60] M. Navardi, B. Ranjbar, N. Rohbani, A. Ejlali, and A. Kumar, "Peak-Power Aware Life-time Reliability Improvement in Fault-Tolerant Mixed-Criticality Systems," *IEEE Open Journal of Circuits and Systems*, vol. 3, pp. 199–215, 2022.

[61] S. Dave, A. Marchisio, M. A. Hanif, A. Guesmi, A. Shrivastava, I. Alouani, and M. Shafique, "Special Session: Towards an Agile Design Methodology for Efficient, Reliable, and Secure ML Systems," in *2022 IEEE 40th VLSI Test Symposium (VTS)*, 2022, pp. 1–14.