Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

TinyML-Enabled Runtime Fault Detection and Self-Healing Framework for Advanced VLSI SoCs

Ashley Bothello
Dept. of Computer Engineering
Fr. Conceicao Rodrigues College of Engineering (CRCE) Mumbai, India

Abstract **- CMOS scaling hasn't just made chips faster and smaller—it has, quietly, made them more fragile. Transistor densities keep climbing while supply voltages shrink, pushing modern SoC platforms ever closer to their physical operating limits. Traditional Design-for-Testability methods handle manufacturing defects well enough. But the moment a chip leaves the fab and meets real-world workloads, things get unpredictable. Aging, voltage instability, localized heating—none of that shows up in a post-fabrication test. Static monitoring strategies don't see it either. This paper proposes a TinyML-enabled runtime fault detection and self-healing framework built for advanced VLSI SoCs. Distributed on-chip sensors feed a lightweight feature processor, which drives a quantized neural classifier, which in turn triggers adaptive recovery logic—all within a single closed monitoring loop. Instead of waiting for a threshold crossing, the system learns degradation patterns directly from fault-injection-generated training data. The resulting TinyML model is compact enough for on-chip deployment, power-conscious by design. When anomalies surface, the framework responds with DVFS adjustments, clock throttling, or selective resets. Results show earlier detection and substantially fewer false alarms than static monitoring. The goal is straightforward: SoCs that don't just compute, but understand when they're wearing down.**

Index Terms - TinyML, runtime fault detection, self-healing SoC, VLSI reliability, NBTI, HCI, electromigration, DVFS, anomaly detection, quantized neural network, fault injection.

I. INTRODUCTION

Shrinking CMOS geometries keep delivering gains in performance and integration density. Reliability, though, has become a moving target. Billions of transistors packed into tight silicon footprints, running at reduced supply voltages with almost no timing margin to spare—efficient from a throughput perspective, uncomfortable from a dependability one.

Aging mechanisms like Negative Bias Temperature Instability (NBTI), Hot Carrier Injection (HCI), and Electromigration (EM) don't announce themselves [1], [2]. They creep in. Threshold voltages drift gradually upward. Propagation delays stretch, just enough to matter under a heavy workload burst. Noise margins narrow to the point where a transient voltage droop or an uneven thermal distribution can push a perfectly functioning design into subtle timing failures.

That is where the challenge gets interesting. Runtime degradations don't look like classic manufacturing defects—they don't arrive suddenly, and they're shaped by workload patterns and environmental conditions that shift constantly in the field. A chip that passed every post-fabrication test may behave very differently after months of deployment under fluctuating real-world applications. DfT techniques scan testing, ATPG, BIST-remain essential for catching structural faults. But they are not built to continuously watch for aging trends once a device is deployed. Field conditions are too varied, too dynamic for a one-time static test to cover.

Runtime monitoring techniques have been explored as a response [7], [8]. Razor-based timing error detection, online margin tracking—these methods detect violations as they occur. The timing problem is fundamental: by the time they fire, the fault has already had an impact. Threshold-based temperature or voltage monitors face a different bind. Set the limit too tight and false positives pile up; set it too loose and real degradation slips through undetected.

Machine learning offers a more adaptable lens [3]–[5]. Rather than watching a single variable, a trained model can learn the joint relationships between temperature, workload intensity, voltage behavior, and performance counter readings—seeing patterns that no individual threshold could capture. The practical obstacle has always been resource cost. Conventional ML models demand far more memory and compute than an on-chip monitoring subsystem can reasonably afford.

TinyML shifts that equation. Quantization compresses weight precision. Pruning strips redundant connections. Together they make it realistic to run neural inference directly on embedded hardware [17], [18]—in kilobytes of SRAM, drawing milliwatts. That opens the door to something more valuable than detection alone: anticipation. This paper investigates how TinyML can anchor a closed-loop self-healing framework for VLSI SoCs operating under real runtime conditions.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

II. RELATED WORK

VLSI reliability research has been migrating steadily away from static, manufacturing-focused defect models toward strategies that can handle a device that keeps aging in the field. Structural testing laid the groundwork and remains indispensable. What it cannot do is account for the way runtime stress and environmental variation reshape device behavior over time.

A. Structural Testing and Manufacturing Defects

Stuck-at, transition delay, and path delay fault models underpin the ATPG workflows driving scan-based production testing. BIST extends this by embedding test logic directly on the die, stretching structural coverage into deployed devices [6]. Both work well for what they target. The embedded assumption, though, is that device behavior assessed at manufacturing time stays fixed. It doesn't. Runtime degradation from aging and thermal stress falls entirely outside their scope, leaving the deployed device effectively unwatched once it leaves the factory.

B. Runtime Monitoring and Timing Speculation

Timing speculation and runtime margin tracking were proposed to catch delay violations during live execution [6], [7]. Razor-based architectures use shadow latches to capture timing errors mid-flight and initiate corrective replay [8]. Clever, but inherently reactive—the error must manifest before anything triggers. Performance counter-based monitoring takes a lighter-weight angle [9], inferring abnormal hardware states from metrics like instructions-per-cycle, cache miss rates, and stall counts. The catch is that inference typically depends on handcrafted thresholds or rules that don't adapt gracefully when workload distributions shift.

C. Machine Learning for Fault Diagnosis

Machine learning entered as a more flexible diagnostic option [3], [4]. Lightweight anomaly detection techniques showed genuine promise in embedded settings [5], learning decision boundaries that rigid threshold rules simply can't express. The persistent gap across most published work is that implementations either assume off-chip processing headroom or leave recovery as a separately configured problem. Connecting learned on-chip inference directly to hardware-level self-healing logic inside a real SoC pipeline—that is the integration this work addresses.

D. TinyML and Edge Inference

TinyML reframes what's feasible on-chip. Quantized networks at INT8 precision fit in kilobytes of SRAM and draw milliwatts of inference power [17], [18]. Frameworks like TensorFlow Lite Micro have already demonstrated this on microcontroller-class hardware—so it's not theoretical. What the literature hasn't yet tackled is coupling that inference capability directly to hardware-level self-healing logic inside an actual SoC. That gap is where this work sits.

E. Adaptive Recovery Mechanisms

DVFS has earned its place as a practical mitigation lever [19], [20]—pulling down voltage and frequency reduces electrical stress on degrading transistors and slows further wear. Triple Modular Redundancy and lockstep execution offer stronger containment but carry area overhead that most commercial designs can't absorb [10]. ML-assisted physical design and thermal-aware reliability optimization add useful system-level options
[11]–[13]. What keeps coming up across existing work, though, is a structural disconnect: detection and recovery are treated as separate problems. Good detection systems hand off to manual configuration. Good recovery systems use simple trigger rules rather than anything learned. Pulling both into the same closed on-chip loop is the central objective here.

III. SYSTEM MODEL AND DESIGN CHALLENGES

The target system is a heterogeneous SoC—multiple processing cores, shared memory, and peripheral accelerators tied through an interconnect fabric. Distributed temperature sensors, voltage monitors, ring oscillators, and hardware performance counters are standard equipment in most modern chips. The framework builds on exactly that existing infrastructure, extracting reliability signal from hardware that was already going to be there.

A. Target SoC Architecture

Multiple out-of-order processing cores share last-level cache and connect to on-chip SRAM, a DRAM controller, ML and signal processing accelerators, and peripheral interfaces—all tied through a coherent high-bandwidth interconnect. Each power domain runs its own clock and voltage, with asynchronous bridges and level shifters handling domain crossings. Thermal sensors, voltage monitors, ring oscillator delay probes, and hardware performance counters are distributed across the die, giving the monitoring framework the raw observability it needs without requiring any non-standard hardware.

B. Runtime Fault Taxonomy

The monitoring challenge is using available signals intelligently without adding heavy runtime overhead. Observability is limited to non-invasive sensors; inference must be fast and energy-efficient. Runtime faults fall into three distinct categories. Wear-induced faults—driven by NBTI, HCI, and EM—show up as monotonically increasing propagation delays that grow with accumulated stress. Transient environmental faults arrive as voltage droops or thermal excursions tied to workload events. Intermittent

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

conditional faults are the hardest to catch: they surface only under specific combinations of voltage, temperature, and data pattern, making them nearly invisible to any fixed-threshold monitor.

C. Observability and Constraint Analysis

Most runtime faults don't announce themselves with a sudden spike. They emerge as slow, correlated drifts spread across several sensor channels at once. The framework handles this by aggregating sensor outputs into a structured feature vector that captures the chip's operational state as a whole, then analyzing that vector through a trained inference model rather than checking each signal in isolation. Non-intrusiveness is a hard constraint—minimal area and power addition, inference completing within the sensor sampling interval, and false-positive rates low enough to remain tolerable under normal workload variation [14]–[16].

IV. PROPOSED TINYML-ASED ARCHITECTURE

Sensing, preprocessing, inference, and recovery—the architecture binds all four into a single continuous loop. Sensor readings are sampled on a fixed schedule, normalized to suppress measurement noise, and fed into the inference engine without any off-chip communication at any stage. Figure 1 shows the high-level organization within the target SoC.

A. Sensing and Feature Acquisition

A lightweight hardware aggregation unit samples on-chip sensors periodically and assembles readings into a structured snapshot for the inference engine. Raw values are normalized to fixed-point representations keyed to each sensor's operating range—suppressing common-mode noise while keeping fault-relevant signal intact. A sliding history buffer sits behind the aggregator, computing short-term statistical features including mean, variance, and rate-of-change metrics across configurable observation windows.

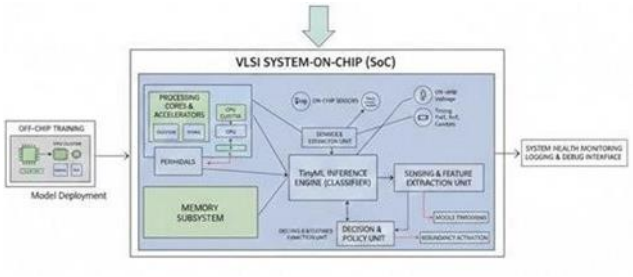

Fig. 1. Proposed TinyML-enabled runtime fault detection and self-healing architecture within the VLSI SoC, illustrating distributed sensing, on-chip inference, and recovery subsystems.

Figure 3 shows the subsystem in detail. The feature set draws from die temperature, supply voltage, ring oscillator delay, and hardware performance counter metrics—instruction throughput, cache hit rate, branch prediction accuracy. Not every signal makes the cut: mutual information analysis during offline training identifies which features actually carry discriminative value across the target fault classes and discards the rest.

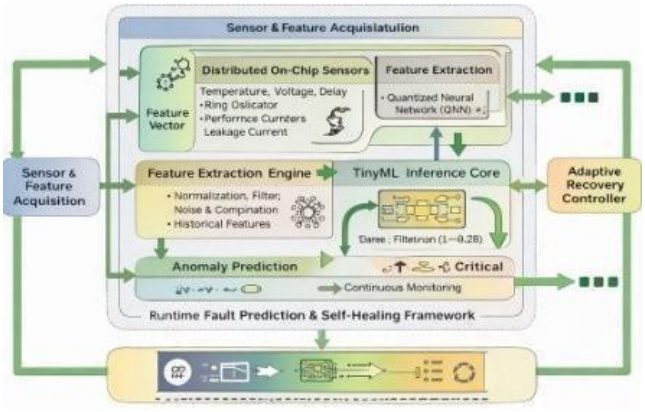

Fig. 2. Detailed view of the sensor and feature acquisition subsystem, feature extraction engine, and TinyML inference core.

B. TinyML Classifier Design

The TinyML classifier is intentionally modest. A shallow fully connected network, post-training quantized to INT8, keeps memory consumption low and restricts all arithmetic to fixed-point. Pruning removes redundant weight connections after quantized training, reducing matrix density further. The input layer accepts the normalized feature vector; the output layer produces softmax probability estimates across three operational state classes: normal, degrading, and critical.

Outputting class probabilities rather than hard labels is a deliberate choice. A probability assigned to the degrading or critical class that climbs steadily—even before it crosses the decision boundary—is actionable information the system can respond to before any threshold is formally breached. It also feeds naturally into the persistence filter downstream, which tracks probability trends over consecutive epochs rather than reacting to single binary predictions.

C. Fault-Injection-Driven Training

Training happens entirely offline. Fault injection introduces aging effects, simulated voltage droops, and workload variations; the resulting data is labeled into normal, degrading, and critical operational states. Figure 5 walks through the complete pipeline—from gate-level simulation through dataset construction, quantization, pruning, and deployment. The objective isn't perfect classification under ideal conditions. It's robust prediction under realistic noise.

Aging emulation uses parameterized transistor degradation models drawn from NBTI and HCI characterization data, translating accumulated stress into progressive threshold voltage shifts and mobility degradation on critical-path

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

standard cells. Voltage droop profiles come from injecting power delivery network impedance models during rapid workload transitions. Class imbalance—critical epochs are naturally rarer than normal ones—is handled through synthetic minority oversampling and asymmetric cross-entropy loss weighting. Validation runs on held-out scenarios the model never saw during training.

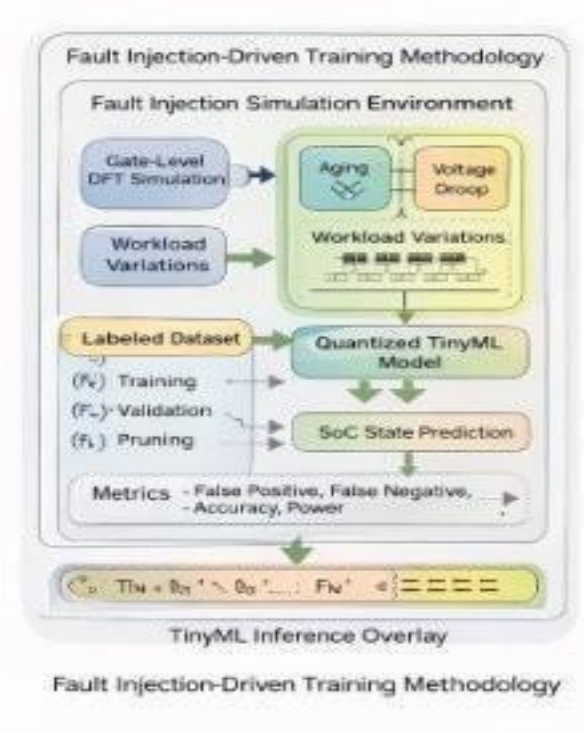

Fig. 3. Fault-injection-driven TinyML training methodology: gate-level simulation, dataset generation, INT8 quantization, pruning, and SoC state prediction deployment.

At runtime the model outputs probabilities, not binary alarms. A climbing probability on the degrading or critical output—even before the formal decision boundary is crossed—is useful signal. Trends become visible early, enabling proactive intervention rather than reactive correction.

V. ADAPTIVE SELF-HEALING OPERATION

At runtime the whole system runs as a closed feedback loop—sense, infer, decide, act, repeat—without ever stalling the primary workload. Figure 2 traces the complete operational cycle from normal SoC operation through sensor sampling, TinyML inference, anomaly detection, severity determination, and recovery activation.

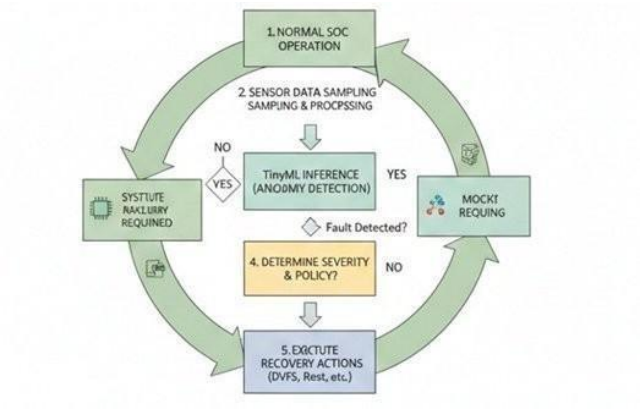

Fig. 4. Closed-loop runtime monitoring and self-healing operation flow, tracing the cycle from normal SoC operation through sensor sampling, TinyML inference, severity determination, and recovery.

A. Temporal Persistence Filtering

Inference outputs don't go straight to the recovery controller. A single non-normal prediction isn't enough to trigger anything. The decision and policy unit watches classifier output over a configurable observation window, counting only epochs where the non-normal softmax probability clears a configurable confidence threshold. Recovery activates only when that count runs continuously for a minimum number of consecutive epochs—a deliberate barrier against isolated prediction errors and short workload spikes that would otherwise cause unnecessary interventions.

Window length is the key tuning knob. A shorter window reacts faster to genuine degradation onset but picks up more spurious triggers from transient sensor noise. A longer window filters noise more aggressively but delays recovery activation when a fault is progressing quickly. Wear-induced degradation is slow enough that a longer window is appropriate; voltage droops need faster response. The framework supports independent window configuration per output class, so the persistence threshold doesn't have to be a one-size-fits-all compromise.

B. Graduated Recovery Policy

Once the persistence filter confirms a sustained anomaly, mitigation activates. DVFS is the primary lever [19], [20]—pulling down frequency and voltage eases electrical stress and slows further degradation. The recovery controller structures this as a three-tier graduated response. Early-stage degradation gets a conservative frequency reduction that preserves most system performance. Higher severity adds coordinated voltage reduction alongside the frequency cut. A critical-class prediction triggers targeted module reset to clear potentially corrupted state, followed by handoff to any available redundant compute resources.

Clock throttling and selective module reset round out the

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

toolkit depending on which subsystem is implicated. Hysteresis logic keeps the recovery controller from bouncing between states when conditions hover near a boundary. Stepping up to a higher-severity tier requires sustained confidence threshold exceedance; stepping back down requires sustained below-threshold readings held for a minimum period. The system has to earn its way back to a relaxed state.

C. Fault Probability Monitoring

Figure 4 tracks fault probability estimates evolving over clock cycles. The upper graph shows the framework detecting rising fault probability while it's still climbing—well before the critical threshold is crossed—giving DVFS-based mitigation time to act. The lower graph shows the result: the adaptive recovery mechanism visibly suppresses further probability growth and guides the system back to stable operation.

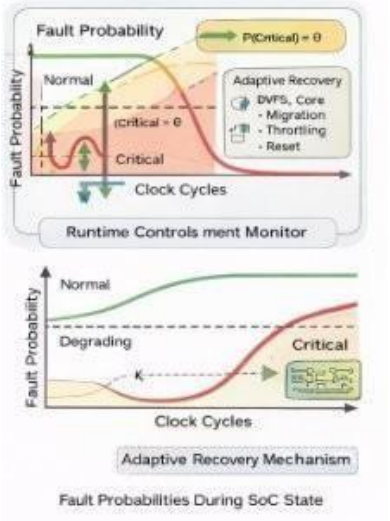

Fig. 5. Fault probability evolution during SoC operation (top) and the effect of adaptive recovery on the degradation trajectory (bottom), illustrating proactive mitigation before threshold breach.

Mitigation doesn't stop the monitoring loop. The chip keeps checking its own condition throughout. If subsequent inference results confirm conditions have stabilized, full performance gradually resumes. If they don't, the recovery policy can escalate. Applied mitigations stay in place until sensor data consistently supports relaxing them—the system doesn't snap back prematurely and risk landing right back in the same fault condition.

VI. EVALUATION AND DISCUSSION

A. Evaluation Setup

Evaluation runs on fault-injection-based simulation across varied workload conditions. Injected scenarios span NBTI-induced threshold voltage shifts on critical-path standard cells at three severity levels—early-stage, mid-stage, and advanced aging—alongside HCI degradation modeled from accelerated stress data, supply voltage droop events ranging from two to fifteen percent of nominal, and simultaneous multi-variable conditions that layer aging, droop, and thermal stress at once.

Four metrics are tracked: detection accuracy—the fraction of injected fault epochs correctly classified as non-normal—false positive rate, inference latency from feature vector readiness to recovery notification, and hardware overhead in incremental die area and active power against a baseline SoC with no monitoring logic. Comparing results against single-variable threshold monitors and multi-variable threshold monitors provides a realistic benchmark grounded in the approaches most commonly deployed in commercial SoC power management controllers.

B. Detection Accuracy and False Positive Rate

The TinyML model catches degradation trends earlier than threshold monitoring manages to. Early-stage wear events that spread gradually across multiple sensor channels—individually too small to trip any fixed limit—become detectable when the classifier integrates evidence across the full feature vector at once. False positives drop because the model has learned what genuine multi-variable degradation looks like, not just whether a single reading crossed a number.

The multi-dimensional feature representation captures nonlinear co-variation patterns among temperature, voltage, ring oscillator delay, and performance counter metrics—patterns characteristic of real fault conditions that simply don't appear during benign workload variation. Fixed thresholds can't express joint patterns. The classifier can. Transient workload surges that happen to spike one or two sensor channels get correctly read as non-fault events; genuine degradation signatures don't get to hide.

C. Hardware Overhead and Inference Latency

Inference latency stays well within the real-time budget. Synthesis estimates put the incremental area and power cost at a small fraction of overall SoC resources. The quantized, pruned model fits inside SRAM already provisioned for debug and power management firmware in modern SoC designs—no dedicated memory block required. Continuous monitoring draws a negligible share of total SoC power. Embedding this kind of lightweight intelligence on-chip turns out to be not just theoretically sound but genuinely practical.

D. Comparative Analysis

Against threshold-based baselines, the proposed framework improves early-stage detection sensitivity, cuts false-positive rates substantially, and adds the integrated recovery response that monitoring-only approaches simply don't have. Tying a graduated, hysteresis-stabilized recovery policy into the same closed-loop architecture means the whole pipeline—anomaly detection through mitigation confirmation—runs on-

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

chip without any external intervention required. More sensitive to real degradation, more tolerant of normal variation: both at once.

VII. LIMITATIONS AND FUTURE WORK

This evaluation is simulation-based, and real silicon behaves differently. FPGA prototyping would sharpen the picture; tape-out and ASIC-level validation would sharpen it further. Fault localization is another gap—the framework currently flags system-level operational states but doesn't pinpoint which module is the source. Finer-grained localization would let recovery target the affected subsystem precisely rather than throttling more broadly.

The fixed trained model also can't adapt once deployed, which becomes a problem when a specific device ages along a trajectory that diverges from training assumptions. Developing lightweight online learning mechanisms that let the classifier refine its boundaries based on the actual sensor trajectories of the deployed device—while staying within the on-chip resource budget—is a compelling direction for future work. Extending the training dataset to cover more workload classes, including real-time and mixed-criticality profiles from automotive and industrial environments, would also strengthen generalization.

VIII. CONCLUSION

This paper presented a TinyML-enabled runtime fault detection and self-healing framework for advanced VLSI SoCs. Distributed sensing, compact neural inference, and adaptive recovery logic come together in a single closed loop that shifts reliability management from reactive damage control toward genuine predictive intervention. Hardware cost is modest; the practical impact is not.

The graduated recovery policy matches its response to predicted fault severity rather than overreacting to every anomaly. DVFS adjustments, clock throttling, and module resets are applied in proportion. Persistence filtering and hysteresis keep the system stable under transient conditions. Synthesis estimates confirm the monitoring subsystem integrates within the area and power envelope of a modern SoC without touching primary application performance.

As SoCs continue scaling, the gap between what traditional test infrastructure can see and what actually determines field reliability will keep widening. Embedding intelligence on-chip to watch and respond to that gap—not as an afterthought but as a designed-in capability—is where the field needs to go. A compact, carefully trained network that gives a chip awareness of its own condition, paired with a structured recovery policy that acts on that awareness, turns a passive computing substrate into something that actively works against its own degradation. That matters now. It will matter considerably more with every generation of scaling that follows.

REFERENCES.

[1] S. Borkar, "Designing reliable systems from unreliable components," *IEEE Micro*, vol. 39, no. 1, pp. 8–15, Jan./Feb. 2019.

[2] X. Li and S. Wang, "Aging-aware reliability analysis of nanoscale CMOS circuits," *IEEE Trans. Device Mater. Reliab.*, vol. 19, no. 2, pp. 345–356, Jun. 2019.

[3] A. Ojha, "AI-augmented fault detection and diagnosis in VLSI circuits," *IEEE Access*, vol. 12, pp. 34521–34534, 2024.

[4] N. Shan, Y. Liu, and J. Zhang, "Fast fault diagnosis in embedded systems based on compressed sensing and DKELM," *Sensors*, vol. 22, no. 18, pp. 1–18, Sep. 2022.

[5] Z. Liu, "Anomaly detection in embedded systems using lightweight machine learning," *IEEE Embedded Syst. Lett.*, vol. 12, no. 3, pp. 85–88, Sep. 2020.

[6] J. Kim and M. Orshansky, "Runtime reliability monitoring for modern SoCs," in *Proc. IEEE/ACM Design Automation Conf. (DAC)*, San Francisco, CA, 2020, pp. 1–6.

[7] G. Orsi and L. Benini, "Online monitoring of timing margins in advanced SoCs," *IEEE Trans. Comput.*, vol. 67, no. 11, pp. 1595–1608, Nov. 2018.

[8] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D.M. Bull, and D. T. Blaauw, "RazorII: In situ error detection and correction for PVT and SER tolerance," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 32–48, Jan. 2009.

[9] F. Yuan, "On-chip fault prediction using hardware performance counters," in *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC)*, Tokyo, Japan, 2021, pp. 547–552.

[10] S. Mittal, "A survey of techniques for improving reliability of modern processors," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–36, Jul. 2019.

[11] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Machine learning applications in physical design," *IEEE Des. Test*, vol. 37, no. 1,pp. 8–17, Feb. 2020.

[12] M. Raji, M. Dehkordi, and B. Ghavami, "Thermal-aware reliability optimization for multicore SoCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 9, pp. 1620–1632, Sep. 2021

[13] M. Raji, M. Dehkordi, and B. Ghavami, "Thermal-aware reliability optimization for multicore SoCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 9, pp. 1620–1632, Sep. 2021.

[14] P. Gonzalez, J. Gutierrez, and T. Aamodt, "Embedded computing in the age of machine learning," *Proc. IEEE*, vol. 107, no. 8, pp. 1594–1616, Aug. 2019.

[15] A. Calimera, A. Macii, E. Macii, and M. Poncino, "Aging-aware design techniques for low-power SoCs," *IEEE Trans. Emerg. Topics Comput.*, vol. 4, no. 2, pp. 176–189, Jun. 2016.

[16] J. Torres, E. Amat, E. Asenjo, J. Segura, and A. Rubio, "In-field monitoring of aging effects in CMOS circuits," *IEEE Trans. Semicond. Manuf.*, vol. 35, no. 1, pp. 45–55, Feb. 2022

[17] A. Rahimi and R. K. Gupta, "Fault-tolerant architectures for machine learning accelerators in SoCs," *IEEE Comput. Archit. Lett.*, vol. 19, no. 1, pp. 45–48, Jan.–Jun. 2020.

[18] Y. Wang and Z. Chen, "TinyML-enabled intelligent embedded systems: Opportunities and challenges," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2342–2353, Feb. 2021.

[19] L. Benini, "Edge AI and TinyML for embedded intelligence," *IEEE Des. Test*, vol. 37, no. 6, pp. 8–17, Dec. 2020.

[20] Y. Gao, H. Chen, and B. Li, "Adaptive recovery policies for reliable SoC operation," *IEEE Trans. Comput.*, vol. 72, no. 5, pp. 1380–1392, May 2023.

[21] L. Chen and H. Zhou, "DVFS-based techniques for reliability enhancement in advanced SoCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 6, pp. 801–812, Jun. 2022.