Special Session: Trustworthy Hardware-AI at the Cloud

(Special Session)

Special Session: Trustworthy Hardware-AI at the Cloud

Francesco Angione[1], Paolo Bernardi[1], Alberto Bosio[2], Harish Dattatraya Dixit[3], Salvatore Pappalardo[2], Annachiara Ruospo[1], Ernesto Sanchez[1], Arani Sinha[4], Vittorio Turco[1]

[1]*Politecnico di Torino, Dip. di Automatica e Informatica, Torino, Italy*
[2]*Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, 69130 Ecully, France*
[3]*Meta Platforms Inc., Menlo Park, CA, USA*
[4]*Intel corporation, Hillsboro, OR, USA*

***Abstract*—Nowadays, AI applications are becoming extremely popular in our everyday life as well as for the industry. Recent incidents involving hyperscalers have revealed that even cloud-based datacenter hardware can experience failures leading to Silent Data Corruptions (SDCs), also called Silent Data Errors (SDEs). This Special Session delves into the implications of such failures on AI workloads, both during training and inference, and explores methodologies for efficiently detecting SDCs or SDEs through dedicated monitoring phases.**

***Index Terms*—artificial intelligence, silent data corruption, cloud, defects, training, inference**

I. Introduction

by Annachiara Ruospo, Arani Sinha

In the last decades, Artificial Intelligence (AI) has definitely revolutionized our daily life, transforming the way we think and design next-generation hardware technologies. This groundbreaking field introduces numerous benefits but, at the same time, it poses new challenges and risks, mainly due to peculiar characteristics of AI models added to characteristics of advanced semiconductor technology. AI models are never 100% accurate, even under ideal conditions (fault-free scenarios): they provide a prediction score that indicates their confidence in performing a specific task, such as classifying an object or segmenting an image. Additionally, current devices are manufactured with the most advanced semiconductor technologies (to achieve performance and reduce power consumption). However, new technologies are more critical in terms of reliability [1]. Newer fabrication technologies present higher failure rates and mainly suffer from multi-bit flips that cannot be detected and corrected by today's error correction codes mechanisms. Moreover, new technologies introduce new defects, a faster aging, and they are more susceptible to external noise (e.g., radiations).

According to recent works in the literature [2]–[4], all these reliability issues may manifest themselves as "silent" errors at the application level; errors that are not captured by error reporting mechanisms within a CPU and hence are not traceable at the hardware level. As an example, an impacted CPU might miscalculate or lose data, and there may be no indication of these errors. These silent errors, formally defined as Silent Data Corruptions (SDC) or Silent Data Errors (SDE) are hard to detect as they are not detectable through conventional Design-for-Testability (DfT) methods including scan-based and array BIST tests [4]. They are ascribed to causes such as manufacturing defects, process marginalities, alpha particles, cosmic rays. It has been experimentally demonstrated that hardware failures such as SDEs can seriously affect the correct behavior of AI hardware devices, resulting in wrong inferences and computations, never-ending neural networks' trainings [5], and months of debug engineering time [2].

Achieving a near-zero failure rate has always been deemed essential for safety-critical systems. However, significant reliability issues are now also observed in large data servers and High Performance Computing (HPC) systems that process vast amounts of data daily. The frequent occurrence of silent errors may corrupt many computations, resulting for example, in substantial data loss. Therefore, it is crucial to devise even more sophisticated and innovative fault-detection techniques to promptly identify hardware defects that have eluded manufacturing tests and could potentially become SDCs or SDEs. However, given the complexity of state-of-the-art AI models (with parameters in trillion range) running on billion-transistor hardware devices, it is highly unfeasible to detect and even to excite the entire fault universe. Next-generation hardware devices should be equipped with novel detection mechanisms primarily focused on identifying faults that may lead to functional failures (e.g., neural networks' wrong predictions). In this light, this research work suggests the use of new fault grading methods that identifies faults that can cause silent data corruption. The research work also proposes an architectural fault-tolerant mechanisms oriented to early detect application failures stemming from hardware defects that have escaped known testing phases and silently propagate through the system stack. To this end, cost-effective monitors may be incorporated into AI hardware accelerators. These monitors continuously monitor the training and inference phases of AI

Authors are listed in alphabetical order.

979-8-3315-2144-8/25/$31.00 ©2025 IEEE

models, tracking *key parameters* that may indicate potential application failures. They also alert the software layer to any deviations in application functionality.

The goal of this article is to discuss the origin of SDCs/SDEs (Section II), their impact on the training (Section III) and their impact on the inference and how it could be possible to reduce the failure rate by introducing monitoring phases for fault detection (Section IV).

II. SDC/SDE in hardware: definition and possible origins

by Francesco Angione, Paolo Bernardi, Arani Sinha

With the advent of AI-based systems, digital systems are increasingly present in everyday life, from IoT applications to data center fleets operated by Cloud Service Providers (CSPs) like Google and Meta [2], [6], [7]. CSPs rely heavily on their semiconductor suppliers' testing strategies to ensure system reliability, serviceability, and availability (RAS) [8], [9].

The growing demand for computational power has led to increasingly complex devices with exponentially high density, pushing testing capabilities to their limits [10]. These challenges are bounded by design complexity, transistor density, test pattern generation time, ATE memory limitations, and DfT constraints [11].

Despite manufacturers employing both structural and functional testing approaches, even achieving 99.00% fault coverage still results in significant test escapes - approximately 200k for a 20M gates device and 10M for a 1B gates device. To address this, manufacturers introduced System-Level Test (SLT) as an additional phase to reduce defective parts per million/billion (DPPM/DPPB) [12].

In always-on data centers, test escapes can manifest as:

I) **Benign faults**: Faults never exercised by the workload or do not cause deviation in behavior.
II) **Correctable faults**: Detected and corrected by mechanisms like ECC.
III) **Detected but uncorrectable faults**: Detected but requiring diagnosis for fault isolation.
IV) **Silent data errors (SDEs/SDCs)**: Undetected errors causing significant revenue loss.

The risk of SDCs increases with aging and depends on implemented RAS mechanisms [13]. This contribution focuses on analyzing time zero undetected faults that could manifest as SDCs, aiming to enable early identification for improved architectural resilience and system reliability.

A. Related Works

SDCs, also known as SDEs, exist in both permanent and transient forms. These errors have garnered significant attention in recent years, particularly in data center environments.

Permanent SDCs primarily manifest as manufacturing test escapes at Time-0, while transient SDCs can arise from aging defects. Recent studies have established a strong correlation between SDCs and small delay faults, which are particularly challenging to detect using traditional scan-oriented approaches but can be more effectively identified through functional testing [14], [15]. The detection of SDCs presents unique challenges in testing environments. These errors manifest under specific combinations of voltage, frequency, temperature, and workload conditions. Additionally, high test iterations are necessary for reliable detection, leading to extended execution times and substantial testing costs. A notable limitation in current testing methodologies is the absence of established quality metrics for functional test programs.

In response to these challenges, major industry players have developed various solutions. Google introduced SiliFuzz, implementing a "fuzzing-by-proxy" approach that utilizes CPU simulators and disassemblers [16]. Meta deployed a dual testing strategy comprising Fleetscanner for out-of-production testing and Ripple for in-production lightweight testing [6]. Alibaba contributed with Farron, a sophisticated SDC mitigation solution incorporating prioritized testing and adaptive temperature control [7].

Major silicon manufacturers have also responded by providing open-source testing frameworks. Intel developed OpenDCDiag [17], AMD created the Open Field Health Check [18], and NVIDIA introduced the Data Center GPU Manager [19]. A significant advancement in this field is the Harpocrates framework [20], which employs genetic generation of specialized test programs and a hardware-in-the-loop approach, supporting various architectures while achieving significantly faster test instruction generation. The authors of [21] present a novel metric and methodology for measuring the vulnerability of different microprocessor components to small delay faults (SDFs), addressing a critical gap in existing reliability metrics that primarily focus on particle strike-induced faults. Through a comprehensive case study on an Ibex RISC-V core, the research demonstrates that vulnerability to SDFs varies significantly across different microarchitectural structures, and traditional protection mechanisms like ECC are less effective against SDFs compared to particle strikes, highlighting the need for specialized protection strategies.

Despite these substantial developments in testing methodologies and frameworks, a comprehensive understanding of which specific manufacturing fault escapes can generate SDCs remains elusive. Current testing strategies predominantly rely on holistic assumptions and limited manufacturer data, indicating a crucial area for future research and development in the field of semiconductor testing and reliability.

B. Limitations of Structural Tests

Structural tests, such as those based on the Scan Chain and Logic/Memory Built-In Self-Test (BIST) architectures, are effective automated approaches for detecting faulty behaviors in integrated circuits. These methods leverage DFT techniques to ensure high fault coverage.

However, Automatic Test Pattern Generation (ATPG) strategies for scan-based testing in complex designs can be computationally expensive, often requiring multi-model incremental generations to achieve the high fault coverage necessary for modern devices [22]. Despite their effectiveness, structural tests have limitations. A significant challenge is the occurrence

of test escapes, where faults undetected by the structural pattern set may manifest in devices. These faults can arise during the operational lifetime of chips, highlighting the inherent limitations of structural testing approaches [23], [24]. Although ATPG aims for comprehensive fault coverage, the generated patterns can sometimes configure the circuit in states that are not representative of its operational mode [25].

Nonetheless, the amount and quality of the ATPG-generated patterns are limited by:

- Targeted fault coverage and ATPG computing time [12].
- ATE memory size for pattern storage [4], [26].
- Complex logic circuit design [23], [27].
- ATE test time per chip [28].
- Design for Testability (DfT) architecture and island partitions [11], [29]–[31].
- The infeasibility of exhaustively testing all possible paths for Small Delay faults due to the massive ATPG efforts required [21].

As a result, the scan-based pattern set is not always exhaustive, leaving gaps in fault detection. These limitations underscore the need for complementary testing strategies, such as functional testing [23].

The ATPG test generation is guided by fault testability, which measures how easily a fault can be excited and propagated to primary outputs. The testability metric focuses on analyzing gate-level netlist characteristics [32]–[35].

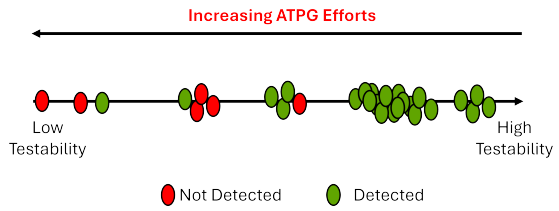


Fig. 1: ATPG Testability oriented fault detection capabilities vs. Test pattern generation efforts.

ATPG efforts primarily detect faults in the higher part of Figure 1. Even if the ATPG algorithm detects low-testability faults, this results in pattern count explosion and not all test content can be accommodated in tester.

Structural test patterns use logic circuits in a non-functional manner. While this approach effectively generates test patterns for covering most faults, it lacks the ability to test functional interactions at speed between logic components, leading to fault escapes during manufacturing, especially for delay-based fault models (transition delay or small delay faults).

Although gate-level netlist-oriented testability measurements can help ATPG develop effective scan-based test patterns [32], [33], they do not adequately analyze which faults, especially test escapes, are more likely to become SDCs in the field. Therefore, examining faults from a different perspective may help guide the generation of functional test programs oriented toward detecting SDCs as fault effects.

C. SDC-oriented Fault Grading Methodology

Traditional testability metrics are based on static circuit analysis, providing for each fault location a value indicating the likelihood of fault detection (excitation and propagation) from that location to the primary outputs. The detection of SDCs has significantly impacted the online testing scenario. The detection and repeatability of SDC occurrences are inconsistent across different functional test iterations, leaving test engineers without a clear measure of functional test program effectiveness. This challenge is compounded by the long execution times of functional test programs aimed at detecting SDCs. Consequently, the development of such programs often relies on test engineers' expertise and holistic assumptions.

This contribution aims to analyze the gate-level netlist of modern silicon devices and identify permanent faults that could potentially lead to SDCs. This goal is driven by the lack of a systematic methodology for functional test programs and the absence of a guided approach to analytically tackle faults. Furthermore, given the broad range of deployable functional test programs, there is no established method for grading these programs' fault detection capabilities, particularly for SDCs. Test engineers typically begin with existing programs that mimic real-world workloads, but since these workloads range from simple matrix multiplication to advanced AI computations, defining test program effectiveness becomes challenging and time-consuming.

The methodology begins by analyzing the ATPG pattern set and introducing test-oriented metrics to differentiate between fault locations with similar circuit-oriented metrics. These metrics are bound to test patterns, as illustrated in Figure 2.

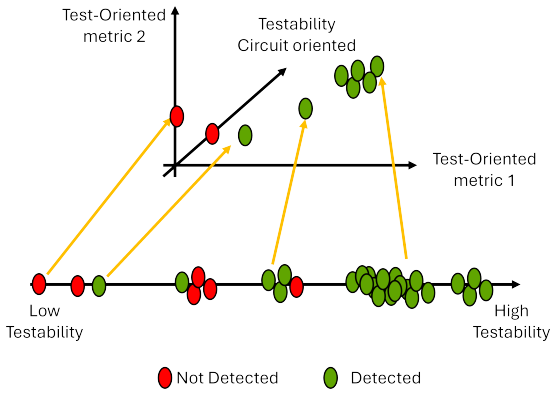


Fig. 2: ATPG Testability measurements (circuit-oriented) combined with different test metrics (test pattern-oriented).

The resolution of test-oriented metrics can be enhanced by providing more fine-grained information, binding the test pattern (whether functional or structural-based) to the logic circuit. This approach of grading faults by combining logic circuit information and test-oriented metrics enables methodology use independent of test pattern type and, to some extent, fault model, as shown in Figure 3. To reduce complexity, the analysis focuses solely on manufacturing test escapes to determine their potential to lead to SDCs in the field.

Module Name	Fault Model	# Faults	Structural tests Fault coverage [%]	# ATPG Test Patterns	ATPG Execution time [h]	Structural + Functional Fault coverage [%]	# Test escapes
Multiplier	SAF	4,959	100.00	34	0.313	100 00	0
	TDF	4,827	99.42	43	4.056	99.42	28
FPU	SAF	71,692	98.59	1,078	2.542	99.40	430
	TDF	67,969	94.94	1,430	8.186	96.09	2658

TABLE I: Experimental results for different modules and fault models.

The methodology provides a comparison between different test patterns in terms of fault characteristics, highlighting faults that may remain unexercised during structural tests but could be exercised by functional programs. Importantly, establishing a threshold for determining SDC-critical faults is necessary, with this threshold depending on environmental factors, customer requirements, and other parameters.

D. Preliminary experimental results

The experimental setup represented in Figure 3 is used to extract circuit-oriented and test-oriented metrics to characterize faults. Testability metrics (SCOAP-oriented [36]) and ATPG test patterns (scan-based) are generated using the *TestMax* tool from *Synopsys*. Functional fault simulations are executed using the *Z01X* tool from *Synopsys*. The activation test-oriented metric is toggle activity, which is extracted from an extended value change dump (eVCD) produced by a logic simulator and analyzed using a custom tool [37].

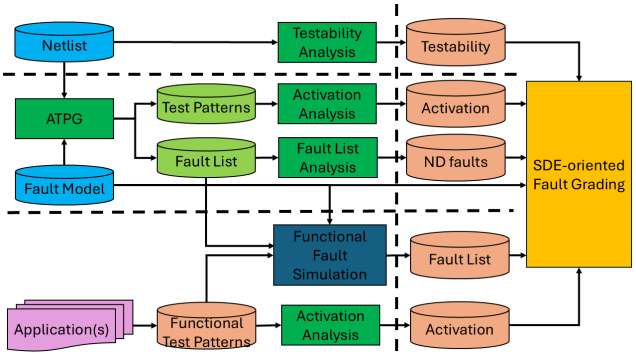


Fig. 3: Workflow for SDC-oriented fault grading methodology.

As Table I shows, experimental results are conducted on two crucial arithmetic modules used in AI workloads: a multiplier and a floating point unit (FPU). SDCs in these modules could negatively affect the outcome of the AI workloads. The modules are synthesized with a clock frequency of 500 MHz and with a technology library of 14 nm.

Table I presents the ATPG coverages, the number of faults, the number of test patterns, and the ATPG execution time for different fault models, such as the Stuck-at Fault Model (SAF) and the Transition Delay Fault Model (TDF). Functional tests, providing randomly generated inputs to the module under test, are added to provide additional fault coverage. For simple modules, such as the multiplier, functional tests do not add any additional fault coverage.

In the last column of Table I, test escapes of structural plus functional tests are presented. With the increasing complexity in terms of the number of faults, the number of test escapes increases, especially for the TDF model. Although structural tests can reach very high coverage for complex modules (FPU), functional tests add an additional fault coverage to structural tests of 0.81% for SAF and 1.15% for TDF.

Starting from the test escapes of structural and functional test strategies, Table II presents the fault characteristics of the undetected faults, trying to spotlight which of them could potentially lead to SDCs if excited and observed during the AI workloads.

Table II presents average values among the test escapes (not detected by strucural tests) of testability measure used to guide the scan pattern generation, and the percentage of activation (toggle activity over the entire simulation) produced by these scan patterns on the test escapes. It compares the structural test escapes' fault characteristics with functional test results by measuring the average percentage activation of additionally detected faults (D), the average percentage activation of undetected faults (ND), and the tenacity (a measure provided by *Z01X* representing fault detection resistance for a given functional test, where higher values indicate harder-to-detect faults).

Transition delay fault escapes for the multiplier unit are highly testable from a structural perspective; however, ATPG did not invest effort in detecting them, as the activation shows. An incremental ATPG run could detect these escapes since the module complexity is low. Regarding the functional test, it is not able to activate or detect any additional faults since the activation of these faults is very low and comparable to that obtained from the structural test patterns. However, from a functional perspective, the tenacity of the undetected faults is relatively low, meaning that a specific functional

Module Name	Fault Model	# Test escapes	Structural		Functional		
			Avg. Testability	Avg. Activation [%]	Avg. Activation D [%]	Avg. Activation ND [%]	Avg. Tenacity ND
Multiplier	SAF	0	NA	NA	NA	NA	NA
	TDF	28	229.95	1.42	0.00	1.26	34.46
FPU	SAF	430	63.04	4.91	13.99	7.18	64.45
	TDF	2,658	55.41	5.40	43.88	6.74	146.60

TABLE II: Test-pattern oriented test escapes characteristics for different modules.

test combined with the existing one could potentially detect such faults. By focusing on a more complex module, the FPU, test escapes are higher. For both SAF and TDF models, from the structural perspective, test escapes are not easily testable even with moderate activation. An additional ATPG incremental run may detect these faults but requires many additional test patterns. Nonetheless, despite the effort that ATPG must invest in detecting these faults, functional tests can detect some of these test escapes, which exhibit more than double the activation compared to structural tests. Moreover, some faults remain undetected even during functional testing, as they activate during the entire simulation but show medium resistance to detection (tenacity). As a consequence, these undetected faults are excited (represented by activation) but not observed.

Therefore, guiding functional test programs to detect SDCs from test escapes can partially exploit tenacity and activation. However, this may not be sufficient to determine which faults have a higher risk of manifesting as SDCs and additional grading techniques need to be developed.

III. SDC/SDE IMPACT ON AI TRAINING WORKLOAD
by Harish Dattatraya Dixit

Artificial intelligence (AI) training has gained significant prominence over the last few years. From deep learning recommendation systems to large-language models (LLMs), a plethora of use cases have been explored across hyperscalers. Even in academia, there is an increasing focus on the evolution and optimization associated with AI models. To train these models, large-scale clusters are designed and built with ever-increasing complexity. Ensuring the reliability of these training models enables lower infrastructure overhead, fewer training iterations, and faster experimentation. Visible faults in the entire cluster domain have a noticeable impact on training effectiveness. Silent Data Corruptions (SDCs) pose even more significant challenges in AI training infrastructures. In this section, we briefly review the aspects of AI training approaches and the impact SDCs have on training applications at scale. We also propose mitigation strategies which could be deployed at various levels of the stack to address challenges related to SDCs.

A. AI training infrastructure

In large-scale infrastructure, different models are trained with different use cases. The model architecture and scale across recommendation systems, language translations, large-language models varies. As a result, the methodologies associated with ensuring reliability, and the implications of failures in such systems varies. Broadly, the training mechanisms are classified into asynchronous training systems and synchronous training systems. In asynchronous training, all the training nodes are not expected to synchronize frequently and objective guarantees are expected to be met at each node, before aggregation. In synchronous training, training nodes operate in tandem and are expected to synchronize and communicate frequently, and achieve training convergence together. Asynchronous training optimizations and efficacy at scale are an active area of experimentation and research. In this paper, we specifically focus on reliability implications in synchronous training workloads.

B. Faults in AI training workloads

In synchronous training, a failure affecting one of the nodes results in the cascaded impact of a training halt for all the nodes in the training cluster. In past papers, a variety of failures have been associated with accelerators in the training clusters. Failures manifest in the computational paths of the AI training nodes, as well as in the communication and synchronization paths.

C. Visible Faults

Visible or *loud* failures are typically associated with multiple sub-domains, the accelerator, the host and the interfaces and communication infrastructure connecting the devices at node level and the cluster level. Failures arise due to bursty correctable and uncorrectable memory errors in local memories, caches, HBMs of the accelerators, or uncorrectable errors in the computational elements, interface and protocol level faults, switch and network related faults. In addition, typical faults associated with the nodes like system memory, system compute and storage faults also have an impact on the training reliability. Furthermore, there are faults in the hardware-software co-design space as well, which provide ambiguous hints to the root-cause of the problem. Any of these faults, on any single node or accelerator directly affect the forward progress of a training workload. Hence, when evaluating the reliability of AI workloads, a cluster level evaluation is crucial for sizing the impact of faults on the overall training efficacy. While these faults are loud, a more insidious variant of faults in training workloads are silent data corruptions.

D. SDCs in training workloads

Silent Data Corruptions (SDCs) in training workloads have drastic implications to training efficacy. SDCs fundamentally result in incorrect computations which are consumed by a training iteration. This incorrect consumption of the corrupted iteration in forward and backward passes in any particular pass, in any accelerator, in any node, creates a divergence in the training path the iteration, as well as the workload, would have taken instead.

It has been asserted that AI training workloads are self-resilient amidst silent data corruptions. However, this assertion holds true in a fairly small subset of the possible SDC manifestations. For example, self-resilience is insufficient in most of the realistic training use-cases and SDC properties. SDCs tend to be persistent throughout many iterations, and AI training workloads pursue quantization of data values representing a higher amount of information per bit; both of which exacerbate the implication of an SDC in a training iteration, and continuously increasing the divergence rates of

training workloads. We present the two common cases of training divergence due to SDCs.

1) NaN Propagation: In the first scenario, we explore NaN propagation. During the computational passes in training iterations, majority of the computations are conducted with fixed point or floating point numerics. These numeric representations are typically range bound on the min and max value representations based on bit-widths. Not-a-Number or NaN are representation of numbers which do not have a valid representation in the specified numeric scheme. As and when an SDC occurs during a computation, the computational error pushes a representable value into a representation that is now incorrect, there by generating a NaN. Furthermore, once a NaN is generated in any part of the training iteration, further computations involving this *NaN* also result in further *NaNs* thereby propagating the NaNs first to the training iteration in flight, then to the accelerator domain, then subsequently to the host domain consisting of many accelerators, and eventually to the cluster domain, which consists of thousands of nodes and many thousands of accelerators. During this process, since the source of NaNs is due to a few specific computations on a single accelerator, which may or may not be individually traceable amidst cluster scale, the cluster has a NaN contagion but the source of the contagion is not traceable. Thus, with SDCs in AI training, the NaN propagations eventually lead to a cluster halt and debug until the offending accelerator and offending nodes are correctly identified and quarantined out of the cluster.

2) Corrupted Gradient Variance: A second and more complex scenario is that of corrupted gradient variance. Gradient variance refers to a gradient explosion or gradient implosion or a local minima because of silent data corruptions. Within the floating and fixed point ranges, a corruption can manifest within the numeric representation bounds. This SDC, while representable, is consumed by the training iteration, and eventually the entire cluster as the *correct* outcome of the training step. However, in synchronous training, a corrupting device with a persistent SDC will continue to corrupt the iteration and have a cascaded impact on the training values across all accelerators within the cluster. The synchronization results in the exchange of corrupted values as *true* values throughout the cluster. As a result, the training will continue to execute but is not guaranteed to make any forward progress as intended by the current state of the hyperparameters. In simple terms, this results in an illusion of forward progress and experimentation, until the result of the SDCs aggregate and result in major divergences to gradients. Depending on the scale, scope, layer and consumption of the corruption, the gradient descent algorithm is stuck at a local minima or the gradients are exploding or imploding beyond the expected ranges. The observability of these SDCs is also dependent on time and computational intensity, and based on the range of the corruptions from true-value, an identification of the existence of a corruption can take weeks if not months. Similar to the previous case, the source of the corruption is not traceable, and is an order of magnitude harder since trapping on NaNs will be insufficient to detect a corruption. So a true SDC, can result in many hours or weeks of cluster computational resource and training iterations rendered unproductive, and the existence of the corruptions without any detection makes the root-cause and subsequent training iterations risky until the offender is quarantined.

E. Impact of SDCs

Presence of SDCs in training clusters results in a cluster level debugging scenario across thousands of components in the entire cluster. While visible faults halt the cluster and require attention to the source of the fault, SDCs result in an illusion of forward progress while hiding the source of the fault. In case of NaN propagation, until the NaN generating node is identified, a training restart from a previous checkpoint without the elimination of the offending accelerator leads to subsequent training runs generating NaNs eventually. Similarly, in case of corrupted gradient variance, the illusion of training progress is persisted longer until the eventual aggregation of the variances repeatedly manifest in subsequent training iterations rendering all the restarts from past known-good checkpoints ineffective. As a result, SDCs can result in significant computational inefficiency at cluster scale having a larger temporal impact than all of the visible faults.

F. Proposed Mitigation Strategies

In this section, we provide an overview of mitigation strategies to deal with SDCs in AI training workloads. We classify them into infrastructure strategies and stack strategies. Infrastructure strategies are applicable in operational triage at the cluster level, while stack strategies require coordination with the workload.

1) Infrastructure Strategies:

- **Reductive Triage**: In cases of NaN propagation, a naive approach is to conduct a binary search with mini-training iterations across continuously reducing cluster sizes. Since the corruption is due to a NaN propagation, the intent with this approach is to arrive at a small enough cluster which replicates a NaN propagation. Upon this replication, the smaller cluster or the offending node, is quarantined for further deeper investigation, while a reconstituted cluster with new nodes is executing training from a previously saved checkpoint. However, this approach assumes a strong reproducibility of the SDCs. Since SDCs are data-dependent, electrical and temperature variation dependent. Even for corrupted gradient variance scenarios, a similar divide-and-triage approach can work, however, the gradient variances and the intentions fundamentally vary with the training input data, as well as the size of the cluster in spite of consistent hyperparameter configurations across all nodes.
- **Fleetscanner and Ripple**: Periodic testing between training restarts and consistently executing a pre-training check on all the nodes with tests enables an alternative screening approach to detecting SDCs. Frequent testing can mitigate SDCs since the mechanisms

for in-production (co-located with training) and out-of-production (between checkpoints) provides near real-time coverage. However, this approach is limited by the coverage of the tests.

- **Deterministic Training**: A known good model can be executed for a few training iterations to validate the absence of NaNs and divergence of gradients. This approach can be used to verify computational failures which are not dependent, since this approach is fundamentally providing a guarantee for a single set of values and training inputs.
- **Hyper Checkpointing**: An alternative method is to checkpoint at increasingly high frequency to allow for accelerated triage of the offending corrupting node, and also allow for lower throughput reduction for the training. In case of NaN propagation, this method can help in containing the NaN to an accelerator or a host, thereby significantly accelerating the triage and quarantine process.

2) Stack Strategies:

- **Gradient Clipping**: A conclusive way to mitigate the NaN propagation scenario for SDCs is to algorithmically enforce gradient clipping as part of the training workload expectations, and bound the training within a certain value range, and any computation that exceeds the range is *clipped* and an example computation generating a NaN could also be detected at the *clipping* step with a *NaN detection* to set to a max-value of a min-value based on the operand sign. However, this potentially works for a subset of NaNs based on representation format, and can sometimes introduce partial error.
- **Algorithmic Fault Tolerance**: A more robust approach is enabling training algorithms with in-built fault tolerance for a range of data corruptions. This would not only reduce the operational burden of detection and triage, but also allow for larger computational efficiency throughout the training workload due to lower overheads. Previously, this approach has been shown to be successful in training on CPUs with minimal overheads. This approach requires understanding the most commonly observed defect modes with engineering investments across the stack and modified guaranatees to the training workloads, at an overhead to the overall training footprint.
- **Tri-Variate Computational Training Architecture**: A key live-approach to mitigate SDCs is with shadow nodes in synchronous training architectures where training steps are repeated across a different set of nodes at random iteration counts. This ensures that the training progress is as intended after a verification step, and helps in cases associated with NaN propagation as well as Corrupted Gradient Variance. Since in both cases, if the shadow nodes, and the live nodes produce different results, the training is immediately halted and only the shadow and live nodes conducting specific training steps are investigated, while the rest of the training continues with new node additions. A tri-variate computational flow performs this operation with multiple shadow node pools and a randomly chosen training node pool of the same size, and specified number of training steps with the same checkpoint origin. This approach provides a robust training infrastructure forward-progress methodology but requires significant algorithmic changes to the cluster trainer and larger data movement and infrastructure overheads.

All the above methodologies for mitigation provide varying levels of resilience with different operating points and similarly different engineering and infrastructure overheads. Based on the scale and the intensity of the training workloads, employing a combination of above strategies can help mitigate the adverse effects of SDCs at scale in AI training.

IV. SDC/SDE IMPACT ON AI INFERENCE WORKLOAD

by Alberto Bosio, Salvatore Pappalardo, Annachiara Ruospo, Ernesto Sanchez, Vittorio Turco

Based on the literature, the recipe to design safe and reliable AI systems includes redundancy as a main ingredient. As an example, a recently introduced international standard, that is, 'ISO / IEC TR 5469: 2024 Artificial intelligence - Functional safety and AI systems' [38], proposes an architectural pattern for systems that use components of AI technology that leverage software redundancy, hardware diversity, combined with a voter mechanism and additional supervisory components and limiting logic units. Redundancy methods are effective in ensuring ultra-low failure rates in hardware units and can help to mitigate SDEs, but they come with area, performance, and power costs. A recent intuition is that it is possible to observe and monitor key parameters in AI systems that can assist the detection of hardware failures, defined as a fault producing an application failure. Paying in terms of time overhead, the authors in [39] propose to rerun suspicious CNN inferences that are potentially corrupted by transient faults, obtaining an Inference Level Parallelism (ILP) based on the confidence of the result. However, completely relying on the output confidence of an inference may be dangerous: they may be highly confident also on wrong predictions. Therefore, while maintaining the concept of utilizing key measures to develop an effective fault-detection mechanism without relying on neural networks' output confidences, our previous works have investigated the fault-detection capability of mathematical metrics typically used for tensors (e.g., [40]). Among the investigated metrics, three were exploited to validate the approach, i.e., single-tensor metrics computing the maximum, the mean, and the sum of all the elements in the tensors (multidimensional arrays corresponding to the output of intermediate layers of artificial neural networks), named Tensor-Max, Tensor-Mean, and Tensor-Sum, respectively. Experimental results computed on two CNNs (ResNet20 and MobileNetV2 on CIFAR10) show that these metrics are able to effectively (i) identify the occurrence of hardware failures on synaptic weights and, moreover, (ii) reveal in advance valuable insights about the final impact a fault might have on the final prediction. They exhibit good properties at differentiating noncritical (i.e., faults

not changing the DNN prediction) from critical faults (also known as SDEs), thus predicting their impact, without waiting for the final output of the inference. The early detection of hardware-induced application failures can save computational time, energy, and power consumption in very deep and large DNNs. Clearly, the deeper the DNN under assessment, the higher the benefit of the approach in terms of costs and energy-power savings. Overall, experimental results indicate that very simple metrics can stop about 96% of SDEs (True Positive Rate - TPR) with a very low False Positive Rate FPR (i.e., an unnecessary early detection mechanism always activated lower than 0.43%).

A. Proposed Fault Detection Methodology

This research work proposes an *early stopping technique* to in-field detect random-hardware faults that propagate as silent data corruption changing the CNN prediction. The approach relies on the computation of tensor-related metrics, applied to the activations (also known as Output Feature Maps - OFMAPS) following convolutional layers. The metrics can act as decision thresholds and trigger a (i) CNN re-execution and (in case of a permanent corruption) the activation of a (ii) safety mechanism. In line with the single tensor metrics proposed in [40], the investigated metrics are the following:

- **tensorMax**: consists in identifying the maximum value of the activations;
- **tensorMean**: it is the arithmetic average of the values of the activations;
- **tensorSum**: computes the sum of all the values of the activations.

The methodology has been validated on two different CNNs running on SAFFIRA, a Systolic Array (SA) fault injector simulator [41]. More in details, if the SA is affected by a fault, through the proposed detection metrics such fault can be detected. A positive detection triggers the re-execution of the computation, the metric is computed again, and another detection is performed. Depending on this procedure, this whole test may return a *pass* (meaning that the fault had a transient effect and has disappeared) or a *critical warning* (the fault is still there). Hence, the SA system enters a halt state (the end-user of the application can decide the safety strategy to adopt). The advantage of computing the metrics without concluding the entire faulty inference (to check the output vector) is obtained in terms of saved FLOPs and saved energy-power consumption. In this preliminary work, the metrics have been computed to the output of the first layer, before the activation function.

The process is carried out in two stages. First, a SA simulation manages the computation of the first CNN layer. Then, the metrics are calculated, and the inference is completed using PyTorch, starting from the second layer and continuing until the CNN outputs are generated. At the same time, architectural-level fault injection (FI) techniques are used to evaluate the effects of faults and validate the method's accuracy and effectiveness. It is important to highlight that metrics computation must occur before the inference is completed to enable online early stopping and fault classification. These metrics can be computed either by modifying the systolic array hardware to calculate them during convolution or by relying on software routines executed by the host processor. This work focuses on the software-based approach, as hardware modifications to the systolic array would require a redesign, which is planned as part of future research efforts.

B. Preliminary experimental results

The proposed technique has been experimentally validated on two CNNs: a ResNet-20 and a MobileNetV2, trained for image classification on CIFAR10, and achieving 91.48% and 91.73% classification accuracy, respectively. Experiments have been executed exploiting SAFFIRA [41]. The designed SA can accommodate the first layer of both CNNs, as it is composed of $32 \times 32 = 1024$ PE. A single matrix multiplication takes $n + 32 + 32$ clock cycles to complete, where n is the length of a row of the input matrix. In order to run the experiment efficiently, we used the same strategy of the one described in [41]. This allowed the inference to be executed in $\sim$1 minute per 100 input images per injected fault.

Transient faults have been randomly injected on the weight register of a random PE of the considered systolic array. Specifically, each fault affects a single bit of the targeted register storing the CNN weight. A statistical subset of 38,500 faulty inferences has been selected (100 random images for each fault), obtaining estimates with an error of 5% and a confidence of 95%. Depending on their effects, injected faults have been classified as critical and non-critical. Among the total faulty inferences (38,500), a total of 2.52% and 2.29% of critical faults have been identified in ResNet20 and MobileNetV2, respectively. The remaining ones have been classified as non-critical. Each register is 32-bit wide and data representation follows the IEEE 754 floating-point standard. The injected fault only affects the computation of the first activations (i.e., the output feature map), so the rest of the inference is assumed to be fault-free.

Thresholds are set at the 100th percentile on the upper end of the distribution. In simpler terms, the maximum metrics value is selected as the threshold for all three metrics. Figures 4a and 4b provide a concise representation of the results. The analysis is performed on all three metrics, but for the sake of space, only the *tensorMax* is illustrated. It is important to note that the general trend is consistent across all metrics. In both figures, the y-axis represents the Probability Density Function (PDF) of the entire distribution of the plotted values, including the metric and other relevant data, while the x-axis corresponds to the range of values for the selected metric, which can be chosen as the threshold for the proposed technique.

First, the PDF of the metric values, computed under the golden scenario across the entire test set (10k images), is plotted (purple histogram). The vertical dashed green lines indicate the maximum values of the tensorMax metric, which are 9.03 for ResNet-20 and 8.16 for MobileNetV2. As previously mentioned, these two values are proposed as thresholds

to trigger the early stopping mechanism and prompt the rerun of suspicious CNN inferences.

The second set of information presented in the same graphs shows the variation of the normalized True Positive (TP) and False Positive (FP) faulty inferences, represented by the blue and red lines, respectively. These curves illustrate the behavior of DNN inferences under varying thresholds conditions. The TP curve represents the trend of inferences that propagate critical faults, which are correctly re-executed, while the FP curve shows the trend of inferences propagating non-critical faults, which are unnecessarily re-executed. Both trends are analyzed with a threshold variation of $\pm 90\%$ along the x-axis. The FP curve serves as an index of the computational cost incurred by applying the fault-tolerance strategy. A comprehensive description of the four confusion matrix indices is provided in Table III.

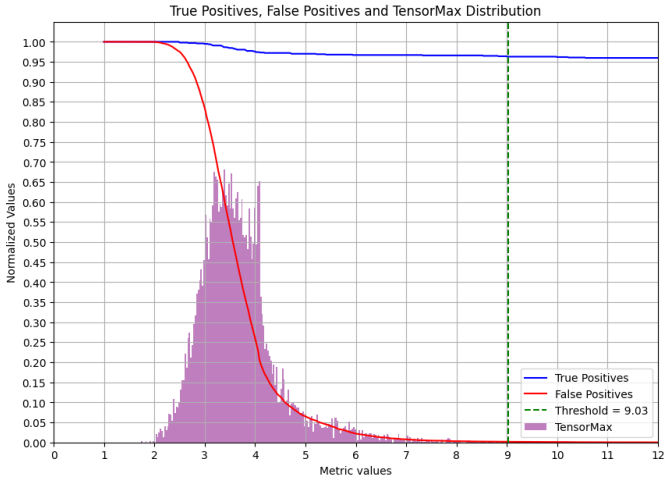
TABLE III: Definitions of the confusion matrix elements.

DNN inferences that propagates critical faults		**DNN inferences that propagate non-critical faults**	
TP	**FN**	**FP**	**TN**
Early detection correctly activated	Early detection wrongly not activated	Unnecessary early detection activated	Early detection correctly not activated

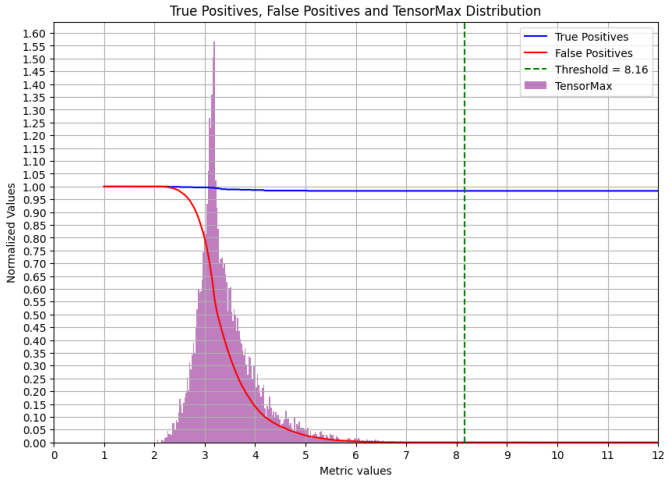
Focusing on the trend of the TP and FP curves on the plots, it emerges that with the two selected thresholds, very high TP values of around 96% are achieved, while maintaining relatively low FP values that discard between 0.25% and 0.43% of non-critical inferences. As the thresholds increase along the x-axis, the changes in terms of TPs are minimal. In contrast, when the thresholds are lowered, both curves show a noticeable slope. For TPs, this results in the identification of more critical inferences. However, it simultaneously causes an exponential increase in the number of FPs, making the threshold computationally expensive and inefficient.

Table IV reports complete results for all the three considered metrics; in all instances, the threshold was determined based on the maximum metric value observed among the fault-free inferences performed to collect the metrics.

It is crucial to emphasize that in this context, the overhead cost is represented by the FP, the noncritical inferences wrongly interrupted. Conversely, the cost in terms of safety and reliability is represented by the FN, the critical inferences that are not interrupted. The results indicate that the TP values are notably high while the FP values remain low, which is a highly positive outcome. Notably, there are minor differences in the metrics for ResNet-20, whereas the metrics for MobileNetV2 exhibit near-identical behavior. Interestingly, the strategy employed for MobileNetV2 results in a TP inference coverage that surpasses that of ResNet-20 by approximately 2.4%, while maintaining similar FP rates. This suggests that MobileNetV2 achieves higher TP rates without a significant increase in FP, demonstrating superior efficiency and reliability.



(a) ResNet20



(b) MobileNetV2

Fig. 4: These graphs illustrate the distribution of the tensorMax values computed over the golden model inferences of the testset for both CNNs.

TABLE IV: Confusion matrix given the selected thresholds.

			Critical Faults		**Non-critical faults**	
Model	**Metric**	**Threshold**	**TP**	**FN**	**FP**	**TN**
Resnet-20	tensorMax	9.03	96.28%	3.72%	0.43%	99.57%
Resnet-20	tensorMean	0.83	95.87%	4.13%	0.26%	99.74%
Resnet-20	tensorSum	13662.50	95.87%	4.13%	0.25%	99.75%
MobilenetV2	tensorMax	8.16	98.28%	1.72%	0.23%	99.77%
MobilenetV2	tensorMean	0.56	98.30%	1.70%	0.25%	99.75%
MobilenetV2	tensorSum	18387.30	98.33%	1.67%	0.24%	99.76%

C. Computational Costs

Calculating the metrics introduces additional overhead to the cost of the inference process. The computational cost of the entire CNN can be quantified in terms of FLOPs, where, in this context, a FLOP is equivalent to a MAC, as it can be executed within a single clock cycle. A standard depth-wise convolutional layer (such as those used in ResNet) requires a total of:

TABLE V: Computational costs of the proposed approach

Details of computational costs	ResNet20	MobileNetV2
Total MACs	$40.56*10^9$	$15.19*10^9$
MACs for computing the first OFMAP	0.44%	0.13%
Shape of the first OFMAP	(32,32,16)	(32,32,32)
MACs for computing the metrics	$16.38*10^3$	$32.77*10^3$
Overhead (%)	$4.03*10^{-5}$	$2.15*10^{-4}$

$$M \cdot N \cdot D_F^2 \cdot D_K^2 \quad \text{MACs},$$

where N and M are respectively the number of input and output channels of the layer, D_F corresponds to the spatial size of the OFMAP and D_K is the spatial size of the kernel (both assumed squared). The cost of the other non-convolutional layers is negligible, as they occupy a small portion of our CNNs. As for MobileNetV2, the implemented convolution is a point-wise separable convolution, which has a different cost. Specifically, its cost is:

$$M \cdot D_K^2 \cdot D_F^2 + N \cdot M \cdot D_F^2.$$

Please note that the cost of a point-wise convolution is much smaller than the cost of the standard depth-wise one. As Table V reports, computing the first OFMAP only occupies about 1% of the total MAC operations of CNN (1. 1% and 0. 83% for ResNet20 and MobileNetV2, respectively). This means that when a faulty inference is correctly stopped (our TP values are always higher than 95%), *the proposed technique can save ∼99% of the FLOPs required to complete that faulty inference.* To compute the metrics, it is necessary to iterate over all elements of the output tensor of the OFMAP, and the cost of computing those metrics is roughly equal to the shape of the tensor. Specifically, tensorMax and tensorSum metrics have the same computational cost. The tensorMean metric has an additional division, accounting for an additional single MAC operation. The table also shows that the overhead produced by computing any of the three metrics is negligible, being on the order of 10^{-4} or 10^{-5} of extra MACs (with respect to the whole CNNs), as given in the last row of the column. Future works will study the trade-off between the number of metrics computations included in more layers of the CNNs and the TP rates of critical faults that are detected and not propagated.

V. Conclusions

This research work explores the possible origins of SDCs/SDEs of a permanent nature, specifically arising from test escapes, since manufacturing tests are not always exhaustive. It provides an SDC-oriented fault grading methodology to highlight test escapes that could potentially manifest as SDCs in the field and guide the development of functional test programs or architectural resilience decisions. Moreover, it discusses their influence on training processes and provides an overview of mitigation strategies to deal with SDCs in AI training workloads. Then, it evaluated the effects of SDCs on AI inference, with a focus on reducing failure rates through the implementation of monitoring phases for fault detection.

Acknowledgments

The study described in Section IV was carried out within the ANR France 2030 AdaptING project “ANR-23-PEIA-0009”.

References

[1] C. Liu, Y. Sun, P. Ren, D. Gao, W. Luo, Z. Chen, and Y. Xia, “New Challenges of Design for Reliability in Advanced Technology Node (Invited),” in *2020 4th IEEE Electron Devices Technology Manufacturing Conference (EDTM)*, 2020, pp. 1–4.

[2] P. H. Hochschild *et al.*, “Cores that don’t count,” in *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 9–16. [Online]. Available: https://doi.org/10.1145/3458336.3465297

[3] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, “Silent data corruptions at scale,” *CoRR*, vol. abs/2102.11245, 2021. [Online]. Available: https://arxiv.org/abs/2102.11245

[4] M. Shamsa and D. Lerner, “Defect mechanisms responsible for silent data errors,” in *2024 IEEE International Reliability Physics Symposium (IRPS)*, 2024, pp. 1–5.

[5] Y. He, M. Hutton, S. Chan, R. De Gruijl, R. Govindaraju, N. Patil, and Y. Li, “Understanding and mitigating hardware failures in deep learning training systems,” in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA ’23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589105

[6] H. D. Dixit, L. Boyle, G. Vunnam, S. Pendharkar, M. Beadon, and S. Sankar, “Detecting silent data corruptions in the wild,” 2022.

[7] S. Wang, G. Zhang, J. Wei, Y. Wang, J. Wu, and Q. Luo, “Understanding silent data corruptions in a large production cpu population,” in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP ’23. New York, NY, USA: Association for Computing Machinery, 2023, p. 216–230. [Online]. Available: https://doi.org/10.1145/3600006.3613149

[8] D. P. Lerner, B. Inkley, S. H. Sahasrabudhe, E. Hansen, L. D. R. Munoz, and A. v. de Ven, “Optimization of tests for managing silicon defects in data centers,” in *2022 IEEE International Test Conference (ITC)*, 2022, pp. 578–582.

[9] S. K. S. Hari *et al.*, “Estimating silent data corruption rates using a two-level model,” 2020.

[10] M. Campbell, “Plenary presentations: Keynote: The product complexity and test — how product complexity impacts test industry,” in *2010 15th IEEE European Test Symposium*, 2010, pp. 9–9.

[11] D. Tille *et al.*, “Towards an automated flow for implementation ofdedicated lbist scan chains for functional safety,” *TUZ*, 2020.

[12] I. Polian, J. Anders, S. Becker, P. Bernardi, K. Chakrabarty, N. El-Hamawy, M. Sauer, A. Singh, M. S. Reorda, and S. Wagner, “Exploring the mysteries of system-level test,” in *2020 IEEE 29th Asian Test Symposium (ATS)*, 2020, pp. 1–6.

[13] D. Trock, S. Mahadevan, N. Mukherjee, L. Harrison, J. Rajski, and J. Tyszer, “Deterministic in-fleet scan test for a cloud computing platform,” in *2024 IEEE International Test Conference (ITC)*, 2024, pp. 391–399.

[14] A. Singh, S. Chakravarty, G. Papadimitriou, and D. Gizopoulos, “Silent data errors: Sources, detection, and modeling,” in *2023 IEEE 41st VLSI Test Symposium (VTS)*, 2023, pp. 1–12.

[15] V. Sridharan, “Addressing emerging fault modes with testing and reliability,” in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems - Keynote*, 2024.

[16] K. Serebryany, M. Lifantsev, K. Shtoyk, D. Kwan, and P. Hochschild, “Silifuzz: Fuzzing cpus by proxy,” 2021.

[17] Intel, “Opendcdiag,” https://github.com/opendcdiag/opendcdiag, accessed: 26 February 2024.

[18] AMD, “Open source field health checks (ofhc),” https://github.com/amd/Open-Field-Health-Check, accessed: 26 March 2024.

[19] NVIDIA, “Data center gpu manager (dcgm),” https://github.com/NVIDIA/DCGM, accessed: 26 March 2024.

[20] N. Karystinos, O. Chatzopoulos, G.-M. Fragkoulis, G. Papadimitriou, D. Gizopoulos, and S. Gurumurthi, “Harpocrates: Breaking the silence of cpu faults through hardware-in-the-loop program generation,” in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 516–531.

[21] P. W. Deutsch, V. Q. Ulitzsch, S. Gurumurthi, V. Sridharan, J. S. Emer, and M. Yan, "Delayavf: Calculating architectural vulnerability factors for delay faults," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 231–245.

[22] M. Fujita, N. Taguchi, K. Iwata, and A. Mishchenko, "Incremental atpg methods for multiple faults under multiple fault models," in *ISQED*, 2015.

[23] H. H. Chen, "Beyond structural test, the rising need for system-level test," in *International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, 2018.

[24] D. K. R. Tipparthi and K. K. Kumar, "Concurrent system level test (cslt) methodology for complex system-on-chip," in *2014 IEEE 16th Electronics Packaging Technology Conference (EPTC)*, 2014, pp. 196–199.

[25] C. Hobeika, C. Thibeault, and J. F. Boland, "Illegal state extraction from register transfer level," in *Proceedings of the 8th IEEE International NEWCAS Conference 2010*, 2010, pp. 245–248.

[26] D. Appello et al., "System-level test: State of the art and challenges," in *IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2021.

[27] G. E. Moore, "Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff." *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, 2006.

[28] G. Iaria, P. Bernardi, C. Bertani, L. Cardone, G. Garozzo, and V. Tancorre, "A comprehensive scan test cost model to optimize the production of very large socs," *IEEE Transactions on Computers*, vol. 74, no. 4, pp. 1278–1292, 2025.

[29] F. Angione, P. Bernardi, G. Filipponi, M. S. Reorda, D. Appello, V. Tancorre, and R. Ugioli, "An optimized burn-in stress flow targeting interconnections logic to embedded memories in automotive systems-on-chip," in *2022 IEEE European Test Symposium (ETS)*, May 2022, pp. 1–6.

[30] F. Angione, P. Bernardi, N. di Gruttola Giardino, G. Filipponi, C. Bertani, and V. Tancorre, "A system-level test methodology for communication peripherals in system-on-chips," *IEEE Transactions on Computers*, pp. 1–8, 2024.

[31] N. Karimi, K. Chakrabarty, P. Gupta, and S. Patil, "Test generation for clock-domain crossing faults in integrated circuits," in *IEEE DATE*, 2012, pp. 406–411.

[32] S. Roy and V. D. Agrawal, "Unsupervised learning provides intelligence for testing hard to detect faults," in *2024 IEEE International Test Conference (ITC)*, 2024, pp. 11–15.

[33] ——, "An amalgamated testability measure derived from machine intelligence," in *2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID)*, 2024, pp. 696–701.

[34] Y. Zeng and X. Cui, "A high performance podem algorithm with the improved backtrace process," in *2024 IEEE International Test Conference in Asia (ITC-Asia)*, 2024, pp. 1–6.

[35] T. Kirkland and M. Mercer, "Algorithms for automatic test-pattern generation," *IEEE Design Test of Computers*, vol. 5, no. 3, pp. 43–55, 1988.

[36] L. Goldstein and E. Thigpen, "Scoap: Sandia controllability/observability analysis program," in *17th Design Automation Conference*, 1980, pp. 190–196.

[37] F. Angione, D. Appello, P. Bernardi, A. Calabrese, S. Quer, M. S. Reorda, V. Tancorre, and R. Ugioli, "A toolchain to quantify burn-in stress effectiveness on large automotive system-on-chips," *IEEE Access*, pp. 1–1, 2023.

[38] ISO and IEC, "Artificial intelligence — functional safety and ai systems," ISO/IEC JTC 1/SC 42, Geneva, CH, Technical Report 5469, 01 2024. [Online]. Available: https://standards.iteh.ai/catalog/standards/iso

[39] A. Mahmoud *et al.*, "Optimizing selective protection for cnn resilience," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, 2021, pp. 127–138.

[40] V. Turco, A. Ruospo, E. Sanchez, and M. S. Reorda, "Early Detection of Permanent Faults in DNNs Through the Application of Tensor-Related Metrics," in *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*, 2024, pp. 13–18.

[41] M. Taheri *et al.*, "SAFFIRA: a Framework for Assessing the Reliability of Systolic-Array-Based DNN Accelerators," in *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*, 2024, pp. 19–24.