

Article

Architecture-Level Risk-Guided Fault-Injection Prioritization for Systolic AI Accelerators: A Fixed Candidate-Pool Evaluation

Larisa Goffman-Vinopal 

Engineering Faculty, Bar-Ilan University, Ramat Gan 5290002, Israel; larisa.goffman-vinopal@biu.ac.il

Abstract

Fault-injection campaigns are widely used to evaluate silent data corruption (SDC) in AI hardware, but exhaustive campaigns over workloads, dataflows, processing elements, and datapath roles are expensive. This paper presents an architecture-level risk-guided fault-injection prioritization method for systolic AI accelerators. The method ranks candidate transient functional perturbations before downstream validation, with the goal of enriching the discovery of candidates that produce a thresholded relative-output-error outcome under a limited validation budget. The evaluation uses a fixed candidate fault pool: all ranking policies score the same 21,000 candidate faults across 30 workload/dataflow/array configurations, corresponding to five GEMM-derived workloads, three array sizes, and two dataflows. Fault magnitudes are sampled once per candidate and are independent of all ranking scores. Candidate faults are modeled as transient architecture-level perturbations in MAC, accumulator, or forwarding paths. The proposed full-risk score combines activity, composite spatial stress, tensor sensitivity, and a path-class weight. In the proposed architecture-level simulation environment and under the fixed-pool protocol, the proposed method achieves the highest mean top-10% SDC-proxy lift, AUPRC, NDCG@10%, and rank correlation with relative output error among the evaluated principle-based ranking policies. At the calibrated threshold, it achieves a mean top-10% lift of $5.65 \times$ [4.91, 6.38], compared with $4.61 \times$ for AVF-like exposure and $4.33 \times$ for output sensitivity. Paired configuration-level tests, threshold sensitivity, and outcome-model sensitivity analyses characterize the result while showing that the proposed score is not universally dominant under every synthetic outcome assumption. The method is intended as a front-end architecture-level screening tool for validation prioritization, not as a replacement for RTL, gate-level, FPGA, or silicon reliability signoff.

Keywords: AI accelerators; systolic arrays; fault injection; silent data corruption; hardware reliability; architectural vulnerability; DNN reliability; reliability screening; validation prioritization



Academic Editor: John A. Kalomiros

Received: 31 May 2026

Revised: 19 June 2026

Accepted: 23 June 2026

Published: 25 June 2026

Copyright: © 2026 by the author.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Systolic-array accelerators are widely used for matrix-intensive AI inference because they exploit regular data movement, high local reuse, and scalable multiply-accumulate throughput [1–6]. Their reliability is increasingly important because local numerical perturbations in multiply-accumulate datapaths, accumulators, or forwarding networks can propagate into incorrect model outputs without producing explicit error signals. Such silent data corruption is difficult to characterize exhaustively because the validation space spans workloads, dataflows, array sizes, spatial locations, datapath roles, and perturbation

magnitudes. Architecture-level activity and mapping analyses further motivate workload- and dataflow-aware screening descriptors [7,8].

Fault injection is a standard method for evaluating reliability, but random campaigns can be inefficient when the number of candidate fault sites is large [9–18]. For systolic AI accelerators, the relevant question is not only whether a local fault can occur but which candidate faults should be prioritized under a limited validation budget. A practical screening method should enrich the discovery of SDC-producing candidates without changing the injected-fault severity distribution or relying only on internal ablation baselines.

This paper studies risk-guided fault-injection prioritization as an architecture-level screening problem. The goal is not to replace detailed RTL simulation, gate-level fault injection, FPGA prototyping, or silicon measurement. Instead, the proposed method is designed as a front-end prioritization layer that ranks candidate faults before more expensive downstream validation. The central question is whether a composite cross-layer score can identify a higher concentration of SDC-producing candidates than simpler principle-based rankings under a fixed candidate-pool protocol.

The fixed-pool protocol is essential. All methods rank exactly the same candidate faults, and each candidate has a perturbation magnitude sampled once independently of all scores. The comparison therefore evaluates ranking quality rather than sampling bias or fault-severity bias. The evaluated baselines are principle-based rankings inspired by established methodological families: systolic activity/utilization modeling [7,8], architectural vulnerability-factor-style exposure analysis [9,10], DNN tensor-level fault-injection practice [13,14], and output-sensitivity ranking. They are not claimed to be complete reimplementations of external tools.

The main contributions are (i) a mathematically specified architecture-level risk score for fault-injection prioritization; (ii) a fixed-pool evaluation protocol using 21,000 common candidate faults across 30 configurations; (iii) explicit definitions of the feature maps, candidate-pool protocol, candidate fault tuple, perturbation magnitude, operational outcome proxy, and threshold; (iv) paired statistical comparisons and threshold sensitivity; (v) an outcome-model sensitivity analysis that explicitly examines dependence on synthetic outcome assumptions; and (vi) an interpretation of how the method can serve as a front-end prioritization layer before more detailed validation.

2. Related Work

2.1. Systolic AI Accelerators and Activity Modeling

Systolic arrays were introduced as regular architectures for high-throughput computation and have become central to modern AI acceleration [1–4]. TPU-style matrix engines, Eyeriss-style spatial architectures, and early neural-network accelerators demonstrate the importance of dataflow, reuse, and utilization in AI hardware design [2–6]. Simulation tools and data-centric analyses such as SCALE-Sim and MAESTRO motivate the use of activity and utilization as meaningful architecture-level descriptors, even though the present work does not reproduce those full tools [7,8].

2.2. Fault Injection and Architectural Vulnerability

Architectural vulnerability analysis distinguishes raw fault occurrence from faults that become visible at higher levels [9,10]. AVF-style reasoning motivates exposure-based reliability metrics that combine activity with the likelihood that a fault affects architectural state. Software-implemented fault tolerance and quantitative fault-injection studies also show that fault outcomes depend on propagation and masking rather than occurrence alone [11,12]. The AVF-like baseline in this paper is inspired by this exposure principle.

2.3. DNN Reliability and Tensor-Level Fault Injection

DNN reliability studies often inject perturbations into tensors, operators, activations, or intermediate numerical values to assess model resilience [13–17]. TensorFI (v2.0.0) and PyTorchFI (v0.6.0) illustrate high-level fault-injection frameworks for machine learning workloads [13,14], while DNN reliability studies emphasize error propagation from low-level perturbations to application-level outputs [16,17]. The DNN tensor FI and output-sensitivity baselines in this paper are motivated by these practices, but they remain principle-based ranking policies rather than complete reproductions of those frameworks.

2.4. Reliability Stress, Aging, and Numerical-Format Sensitivity

Reliability screening for AI hardware also intersects with thermal, aging, and numerical-format effects. Thermal and dynamic-management studies show that localized activity and temperature can affect hardware margins [19,20], while technology variability and aging motivate conservative design margins and lifetime-aware reliability management [21,22]. At the numerical level, low-precision formats such as FP8 and other 8-bit representations change dynamic range, rounding behavior, and error sensitivity in deep learning workloads [23,24]. These lines of work motivate the present paper's use of activity, stress, tensor sensitivity, and path-class information as architecture-level prioritization features rather than calibrated physical failure probabilities.

2.5. Gap Addressed by This Work

Prior approaches provide useful principles but do not by themselves solve the candidate-prioritization problem for systolic-array AI hardware. Activity-based methods capture utilization but not path-class propagation. AVF-like exposure captures architectural exposure but not tensor sensitivity or spatial stress concentration. Tensor-level fault-injection principles capture functional sensitivity but can miss systolic datapath roles. The proposed full-risk score combines these dimensions and evaluates the benefit using a common fixed candidate-pool comparison.

3. Materials and Methods

3.1. Simulation Environment and Accelerator Abstraction

This study uses an architecture-level systolic-array abstraction implemented in Python with NumPy/Pandas-based analysis. It is not an RTL simulator, a gate-level netlist simulation, or a cycle-accurate reproduction of SCALE-Sim. The abstraction is used to create controlled workload-dependent activity, stress, and sensitivity maps so that prioritization methods can be compared on a common candidate pool. The emphasis is on validation prioritization, not physical signoff. The implementation used Python 3.12.13 with NumPy 2.0.2 and pandas 2.2.2 for numerical computation and data analysis; SciPy 1.16.3 for bootstrap confidence intervals and Wilcoxon signed-rank tests; and Matplotlib 3.10.0 for figure generation. The principal model components and evaluation protocol are summarized in Table 1.

The 30 configurations correspond to the full Cartesian product of five workload shapes, three array sizes, and two dataflows: $5 \times 3 \times 2 = 30$. No configuration subsampling is used. The workload shapes in Table 2 are not intended to reproduce full end-to-end neural-network execution. They provide GEMM-derived activity patterns that resemble common accelerator workloads and are sufficient for a controlled comparison of ranking policies. The hotspot-sparse workload is an intentionally stress-oriented case for spatially concentrated activity, not a claim about a universal workload distribution.

Table 1. Architecture-level simulation environment.

Component	Specification
Array sizes	8×8 , 16×16 , and 32×32 processing-element arrays
Dataflows	Weight-stationary (WS) and output-stationary (OS)
Workloads	Five GEMM-derived shapes: square GEMM, transformer QKV, transformer MLP, convolution-im2col, and hotspot-sparse stress case
Path classes	MAC, accumulator (ACC), and forwarding (FWD)
Candidate fault	$q = (i, j, c, \delta)$: tile location, path class, and perturbation magnitude
Fault type	Transient architecture-level functional perturbation; not a permanent stuck-at fault and not an RTL/gate-level bit flip
Main SDC threshold	Relative output error at or above 0.020
Candidate pool	30 configurations, 700 candidates per configuration, 21,000 total candidates
Evaluation protocol	All methods rank the same candidate pool; magnitude and outcomes are fixed before ranking

Table 2. GEMM-derived workload shapes.

Workload	M	K	N	Purpose
Square-GEMM	256	256	256	Dense control workload
Transformer-QKV	196	768	768	Transformer projection-like shape
Transformer-MLP	196	768	3072	Transformer feed-forward expansion-like shape
Conv-im2col	784	576	128	Convolution lowered-to-GEMM-like shape
Hotspot-sparse	512	1024	512	Structured stress-test workload with concentrated activity

For each configuration, the evaluation uses a stored pool of 700 candidate tuples. The pool, candidate identifiers, perturbation magnitudes, and simulated outcomes are fixed before any policy score is computed; every policy therefore ranks exactly the same tuples. The resulting metrics quantify enrichment within this common screening pool rather than an absolute physical defect or failure-rate distribution.

3.2. Feature-Map Construction

For each workload/dataflow/array configuration, the simulator constructs five normalized tile-level feature maps: activity $A(i,j)$, composite spatial stress $H(i,j)$, tensor sensitivity $T(i,j)$, residency $R(i,j)$, and relay burden $L(i,j)$. These maps are not calibrated physical failure probabilities. They are architecture-level descriptors used to compare ranking policies under a common fixed candidate-pool protocol.

Let the array contain $M_a \times N_a$ processing elements. For each tile (i,j) , let $n_mac(i,j)$ denote the estimated number of active multiply-accumulate operations mapped to that tile, $n_res(i,j)$ the estimated number of mapping steps during which partial sums or stationary operands remain resident, and $n_relay(i,j)$ the estimated number of operand-forwarding events passing through the tile. Each raw map is min-max-normalized within its configuration.

$$\text{norm}(X(i,j)) = [X(i,j) - \min_{\{u,v\}} X(u,v)] / [\max_{\{u,v\}} X(u,v) - \min_{\{u,v\}} X(u,v) + \epsilon].$$

The activity map is defined as $A(i,j) = \text{norm}(n_mac(i,j))$. Thus, $A(i,j)$ captures where computation is concentrated in the systolic fabric. Dense GEMM workloads produce relatively broad activity maps, whereas the hotspot-sparse stress case produces spatially concentrated activity.

The residency map is defined as $R(i,j) = \text{norm}(n_res(i,j))$. This term captures dataflow-dependent value retention. In output-stationary execution, partial sums remain local for longer periods, increasing residency in tiles responsible for output accumulation. In weight-stationary execution, residency reflects the spatial persistence of stationary weights.

The relay-burden map is defined as $L(i,j) = \text{norm}(n_relay(i,j))$. Relay burden captures the degree to which a tile participates in operand movement rather than only local arithmetic. A tile with high relay burden may forward activations or weights to downstream tiles even when its own local arithmetic activity is not maximal.

The composite spatial-stress map combines activity, residency, and relay burden:

$$H(i,j) = \text{norm}(0.55 A(i,j) + 0.25 R(i,j) + 0.20 L(i,j)).$$

This map is a compact architecture-level proxy for spatially concentrated execution stress. It is intentionally correlated with the underlying activity, residency, and relay descriptors rather than an independently calibrated thermal field. It is not a replacement for thermal simulation or signoff timing analysis; it identifies regions where workload mapping and dataflow jointly concentrate switching, local retention, and operand movement.

The tensor-sensitivity map captures whether perturbations at a given tile are likely to affect numerically influential output regions. Let $\Omega(i,j)$ denote the output block influenced by tile (i,j) , and let Y_0 be the golden output. The raw tensor contribution is estimated as the mean absolute golden-output magnitude over the affected output block:

$$g(i,j) = \text{mean}_{\{(m,n) \in \Omega(i,j)\}} |Y_0(m,n)|, \quad T(i,j) = \text{norm}(g(i,j)).$$

This produces a tensor-level sensitivity descriptor analogous to the principle used in DNN tensor/operator fault-injection studies, without claiming to reproduce a complete framework such as TensorFI or PyTorchFI.

All feature maps are computed from the clean workload execution and mapping descriptors before any fault injection is performed. They do not use a candidate-level injected-fault outcome, relative-error value, or SDC label. In particular, the tensor-sensitivity map uses only clean-execution numerical structure, such as golden-output block magnitude, as a workload descriptor. The ranking policies therefore use pre-injection information that would be available before expensive fault evaluation, rather than information from the outcomes on which they are later assessed.

A configuration-level heterogeneity statistic may be reported descriptively as $N_u = \sigma(H)/[\mu(H) + \epsilon]$, where $\mu(H)$ and $\sigma(H)$ are the mean and standard deviation of H over tiles. However, N_u is constant for every candidate within one configuration. Because the reported ranking metrics are evaluated within configuration, multiplying every candidate score by a configuration-constant factor cannot change candidate order, top-k membership, AUPRC, NDCG, or rank correlation. Accordingly, N_u is retained only as a descriptive summary and is excluded from the finalized candidate-level score; this correction leaves the reported ranking results unchanged. Table 3 provides an intuitive interpretation of the pre-injection descriptors.

Table 3. Intuitive interpretation of pre-injection descriptors.

Feature	Intuitive Meaning	Role in Analysis
$A(i,j)$	Where computation occurs	Prioritizes active compute regions
$R(i,j)$	Where values remain resident	Captures persistence of partial sums or stationary operands

Table 3. Cont.

Feature	Intuitive Meaning	Role in Analysis
L(i,j)	Where operands are forwarded	Captures relay and propagation burden
H(i,j)	Where activity, residency, and relay concentrate	Architecture-level spatial stress proxy
T(i,j)	Where output contributions are numerically influential	Tensor/output sensitivity proxy
N_u	How heterogeneous the stress map is globally	Descriptive only; not a candidate-ranking term

3.3. Fixed Candidate-Pool Protocol and Perturbation Model

Each candidate is represented by a tuple consisting of a spatial tile, a path class, and a perturbation magnitude. For each configuration, the 700 tuples are instantiated once, stored, and reused without replacement by every ranking policy. Candidate generation defines the common screening pool; it is not intended to estimate a physical defect-rate distribution. The final comparison is therefore about which policy orders the same stored candidates most effectively under a limited validation budget.

$$q = (i, j, c, \delta), \quad c \in \{\text{MAC}, \text{ACC}, \text{FWD}\}.$$

The perturbation magnitude is sampled once per candidate and then held fixed for all ranking methods. It is independent of the proposed score and all baselines. In the calibrated evaluation, the raw magnitude follows a log-normal distribution and is clipped to avoid unrealistically small or extreme perturbations:

$$\delta = \text{clip}(\exp(z), 0.30, 2.20), \quad z \sim \text{Normal}(0.08, 0.33^2).$$

The path class identifies the functional destination of the architecture-level perturbation: MAC perturbations affect a local arithmetic contribution, ACC perturbations affect partial-sum accumulation, and FWD perturbations affect forwarded operands. The candidate model is an abstract functional perturbation model whose candidate-specific severity enters the simulated outcome through δ ; it is not a device-level cross section, a permanent defect model, an RTL bit flip, or a cycle-accurate force/release mechanism.

3.4. Synthetic Outcome Model and Operational SDC Proxy

For each candidate q , the simulator computes a synthetic relative output-error score $e(q)$. Let A , H , T , R , and L denote normalized activity, composite spatial stress, tensor sensitivity, residency, and relay burden at tile (i,j) , respectively. Let $w(c)$ be the path-class gain. The calibrated balanced outcome model is

$$e(q) = \kappa \cdot \delta \cdot w(c) \cdot \varphi(i,j,c) \cdot \eta, \quad \eta \sim \text{LogNormal}(0, 0.26^2),$$

$$\varphi = 0.30 + 0.45 A + 0.34 H + 0.42 T + 0.13 R + 0.12 L + 0.36 AH + 0.22 TH,$$

$$\kappa = 0.0042, \quad w(\text{MAC}) = 1.00, \quad w(\text{ACC}) = 1.55, \quad w(\text{FWD}) = 1.35.$$

In the absence of an explicit architectural detection model, this study operationalizes an SDC-producing candidate as a candidate whose relative output-error proxy exceeds a threshold:

$$\text{SDC}_{\text{proxy}}(q) = 1\{e(q) \geq \tau\}, \quad \tau = 0.020.$$

The threshold $\tau = 0.020$ is not intended to represent a universal application-level accuracy requirement or a calibrated silicon failure boundary. It is selected as a rare-event operating point that yields a mean base SDC-proxy fraction of approximately 1.8% across configurations. This makes the evaluation a prioritization problem: random selection finds relatively few threshold-exceeding candidates, so ranking quality can be compared meaningfully. To reduce dependence on a single operating point, sensitivity is also reported for τ in $\{0.016, 0.020, 0.024, 0.028\}$.

3.5. Ranking-Score Definitions and Baselines

Each method assigns a scalar ranking score $S(q)$ to every candidate in the fixed pool. Candidates are sorted by decreasing score and evaluated by how many threshold-exceeding or high-error candidates appear near the top of the ranking. All maps are normalized to $[0, 1]$ within each configuration. The baselines are principle-based ranking policies inspired by prior methodological families; they are not full reimplementations of external tools. The formal definitions of the evaluated ranking scores are given in Table 4.

Table 4. Formal ranking-score definitions.

Method	Ranking Score	Interpretation
Random FI	$S(q) = u(q), u(q) \sim \text{Uniform}(0, 1)$	Unbiased random candidate ranking
Uniform-active PE	$S(q) = 1$	Equal priority over active candidate sites
Activity-only	$S(q) = A(i,j)$	MAC/activity-based screening
Systolic activity-utilization	$S(q) = U(i,j) = \text{norm} [0.70 A(i,j) + 0.20 L(i,j) + 0.10 R(i,j)]$	Systolic-array utilization principle inspired by accelerator activity modeling
AVF-like exposure	$S(q) = A(i,j)[0.55 + 0.45 R(i,j)]w_AVF(c)$	Architectural-vulnerability-style activity times exposure
DNN tensor FI	$S(q) = T(i,j)$	Tensor-sensitivity principle inspired by tensor/operator fault injection
Output sensitivity	$S(q) = T(i,j)[0.70 + 0.30 L(i,j)]w_OUT(c)$	Strong heuristic based on estimated output-error propagation
Proposed full-risk	$S(q) = [0.40 + 0.60 A(i,j)][0.35 + 0.65 H(i,j)][0.45 + 0.55 T(i,j)]w_R(c)$	Combined activity, composite spatial stress, tensor sensitivity, and path-class weight

The path-class ranking weights used in the reported simulations are $w_AVF(\text{MAC}, \text{ACC}, \text{FWD}) = (1.00, 1.35, 1.20)$, $w_OUT(\text{MAC}, \text{ACC}, \text{FWD}) = (1.00, 1.45, 1.25)$, and $w_R(\text{MAC}, \text{ACC}, \text{FWD}) = (1.00, 1.40, 1.25)$. These fixed weights encode the qualitative assumption that accumulator perturbations are more persistent than MAC perturbations and that forwarding perturbations can propagate spatially. They are deliberately distinct from the hidden outcome-model path gains, so the ranking score does not reuse those gains verbatim.

The proposed score is intentionally multiplicative: a candidate receives high priority only when several pre-injection risk dimensions align. Within the proposed score, residency and relay burden contribute through the composite stress term H rather than as separate multiplicative factors. This differs from activity-only or AVF-like ranking, which can prioritize highly active regions even when tensor sensitivity or path-class propagation is weak. It also differs from output-sensitivity ranking, which emphasizes functional effect but does not explicitly include composite spatial stress.

For interpretation, the full-risk score is the alignment of four prioritization factors:

$$S_{\text{risk}}(q) = [0.40 + 0.60 A(i,j)] \cdot [0.35 + 0.65 H(i,j)] \cdot [0.45 + 0.55 T(i,j)] \cdot w^R(c).$$

The four factors correspond to activity, composite spatial stress, tensor sensitivity, and path class. The offsets are scaling constants, not calibrated probabilities; they prevent a single zero-valued descriptor from forcing the composite score to zero. All reported

weights are held fixed across configurations. Figure 1 summarizes the pre-injection construction of this score.

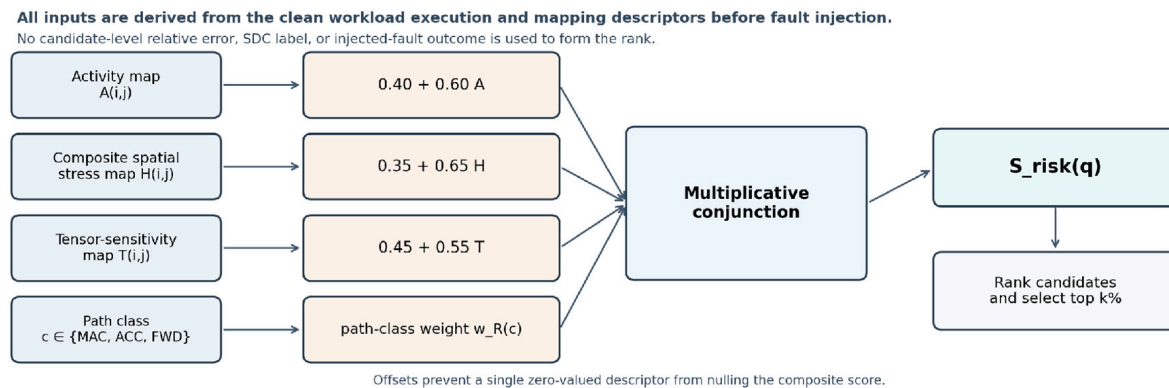


Figure 1. Pre-injection construction of the proposed multiplicative risk score. The rank uses only clean-execution and mapping descriptors; fault outcomes are not used to form the score. Arrows indicate the transformations and dependencies through which each pre-injection descriptor contributes to the final candidate score.

3.6. Metrics and Statistical Testing

Top-k SDC lift is the SDC fraction among the top-ranked k% candidates divided by the base SDC fraction in the full candidate pool. A lift above 1 indicates enrichment of SDC-producing candidates under a fixed validation budget. AUPRC measures ranking quality for rare SDC-producing candidates. NDCG@10% measures whether the top-ranked candidates also have high relative output error. Spearman rank correlation measures monotonic association between score and relative error.

Confidence intervals are computed by bootstrap resampling over the 30 configurations. Paired comparisons between the proposed method and strong baselines use configuration-level paired differences in top-10% SDC lift and one-sided Wilcoxon signed-rank tests. The paired design is important because all methods are evaluated on the same configurations and candidate pools.

3.7. Computational Overhead and Validation-Budget Interpretation

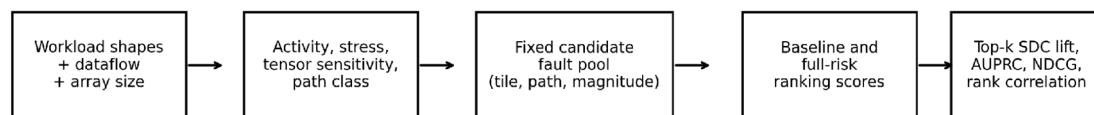
The purpose of risk-guided screening is to reduce expensive dynamic fault-injection effort by ranking candidates before detailed evaluation. The score computation is performed once per configuration and is linear in the number of candidate faults. Spatial maps are computed once per workload/dataflow/array configuration. Candidate scoring then uses constant-time operations per candidate. The ranking step requires sorting or top-k selection over the candidate pool.

$$\text{Cost(screening)} = O(P) + O(C) + O(C \log C),$$

where P is the number of processing elements in the array and C is the number of candidate faults. In the reported evaluation, $C = 21,000$ across all configurations. Selecting the top 10% candidates would reduce a subsequent dynamic validation campaign from 21,000 to 2100 candidates. The intended use case is therefore not to replace fault injection but to prioritize it before more expensive simulation, emulation, RTL, gate-level, FPGA, or silicon validation. Table 5 summarizes the screening and downstream-validation costs; C_{dyn} denotes the number of candidates selected for dynamic validation, and F denotes the average cost per dynamically evaluated candidate. The fixed candidate-pool evaluation flow is shown in Figure 2.

Table 5. Computational-overhead interpretation.

Step	Scaling	Role in Workflow
Activity/stress/tensor map construction	$O(P)$ per configuration	Precomputed once from workload/dataflow descriptors
Candidate score evaluation	$O(C)$	Constant-time scoring for each candidate
Candidate ranking/top-k selection	$O(C \log C)$ or $O(C)$ selection	Produces validation priority list
Dynamic fault injection	$O(C_{\text{dyn}} \times F)$	Expensive downstream step; C_{dyn} can be reduced to top-k candidates



All ranking methods score the same fixed fault candidates; fault magnitudes are independent of ranking scores.

Figure 2. Fixed candidate-pool evaluation flow. All methods rank the same candidate faults; fault magnitudes and outcomes are independent of ranking scores.

4. Results

4.1. Calibration of the SDC Discovery Task

The calibrated outcome model produces a mean base SDC-proxy fraction of 0.018 across configurations, with a minimum of 0.006 and a maximum of 0.036 at the primary threshold. This calibration makes the task a rare-event prioritization problem: random selection finds few threshold-exceeding candidates, so ranking quality is important.

4.2. Main Comparison with Principle-Based Baselines

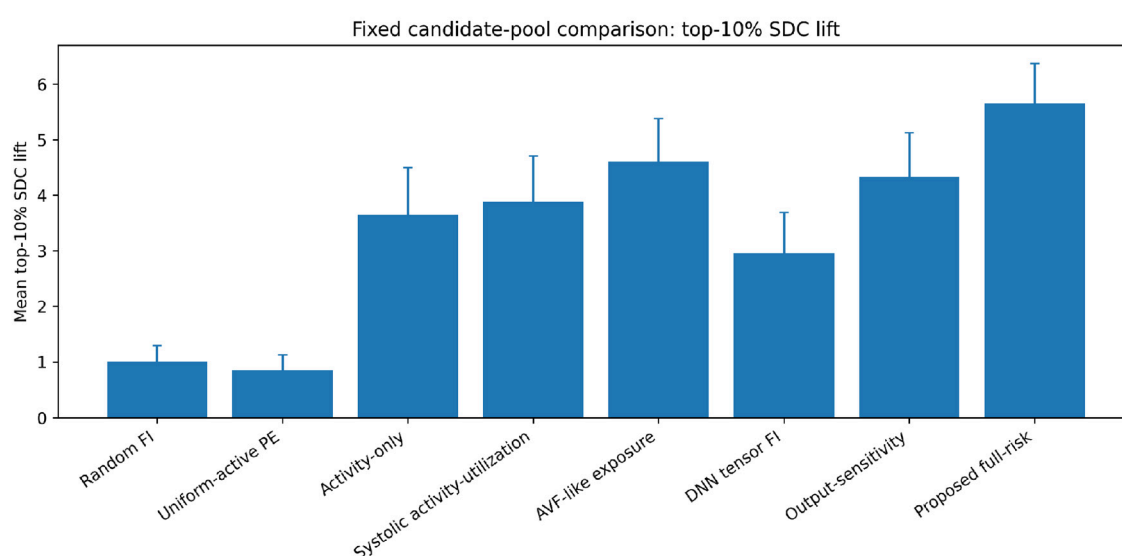
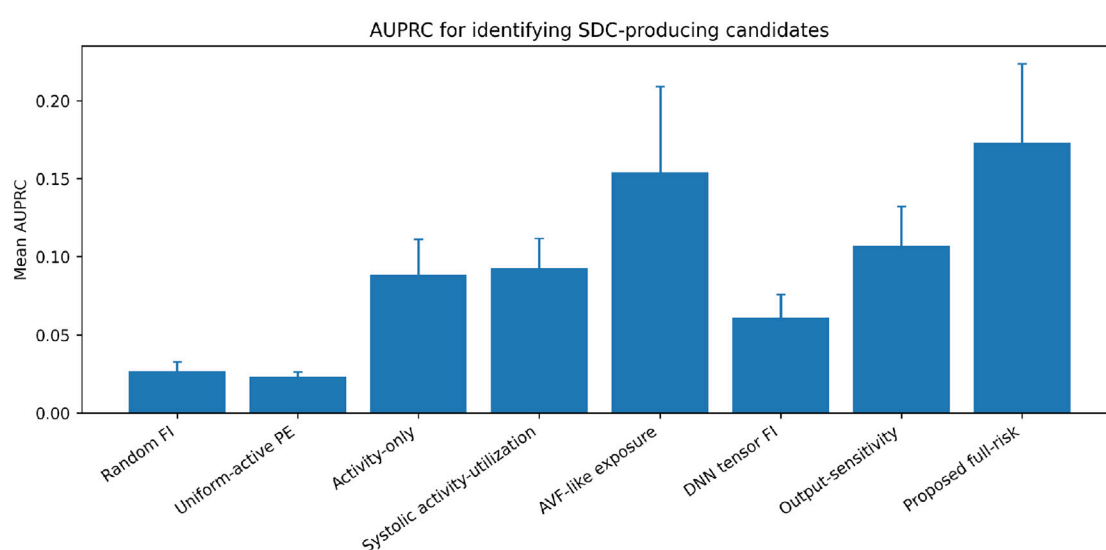
Within the proposed architecture-level simulation environment, the full-risk method achieves the highest mean top-10% SDC-proxy lift, AUPRC, NDCG@10%, and rank correlation with relative error among the evaluated policies. It reaches a top-10% lift of $5.65 \times [4.91, 6.38]$, compared with $4.61 \times$ for AVF-like exposure, $4.33 \times$ for output sensitivity, $3.88 \times$ for systolic activity-utilization, $3.65 \times$ for activity-only ranking, and $2.96 \times$ for DNN tensor FI. Because every policy ranks the same fixed candidate pool, the comparison isolates ordering quality under a shared candidate set rather than a difference in injected-fault distributions. The full comparison is reported in Table 6, and Figures 3–5 visualize the lift, AUPRC, and NDCG@10% results, respectively.

4.3. Paired Statistical Comparison Against Strong Baselines

The paired tests compare the full-risk score with each baseline on the same 30 configurations. The full-risk score improves over AVF-like exposure in 19 of 30 configurations and over output sensitivity in 22 of 30 configurations. One-sided Wilcoxon tests show statistically significant positive paired differences against all listed baselines. The comparison is strongest against activity-only, systolic activity-utilization, and DNN tensor FI, and remains positive against the stronger AVF-like and output-sensitivity baselines. Table 7 reports the paired configuration-level comparisons.

Table 6. Main fixed candidate-pool comparison across 30 configurations.

Method	Top-10% SDC-Proxy Lift [95% CI]	AUPRC	NDCG@10%	Spearman vs. Error
Random FI	1.00 [0.71, 1.29]	0.027	0.389	0.004
Uniform-active PE	0.85 [0.58, 1.13]	0.023	0.393	-
Activity-only	3.65 [2.86, 4.50]	0.088	0.567	0.410
Systolic activity-utilization	3.88 [3.14, 4.71]	0.093	0.585	0.466
AVF-like exposure	4.61 [3.78, 5.39]	0.154	0.641	0.503
DNN tensor FI	2.96 [2.29, 3.69]	0.061	0.542	0.387
Output-sensitivity	4.33 [3.62, 5.13]	0.107	0.628	0.527
Proposed full-risk	5.65 [4.91, 6.38]	0.173	0.685	0.625

**Figure 3.** Mean top-10% threshold-exceedance lift with bootstrap 95% confidence intervals.**Figure 4.** AUPRC for identifying threshold-exceeding candidates.

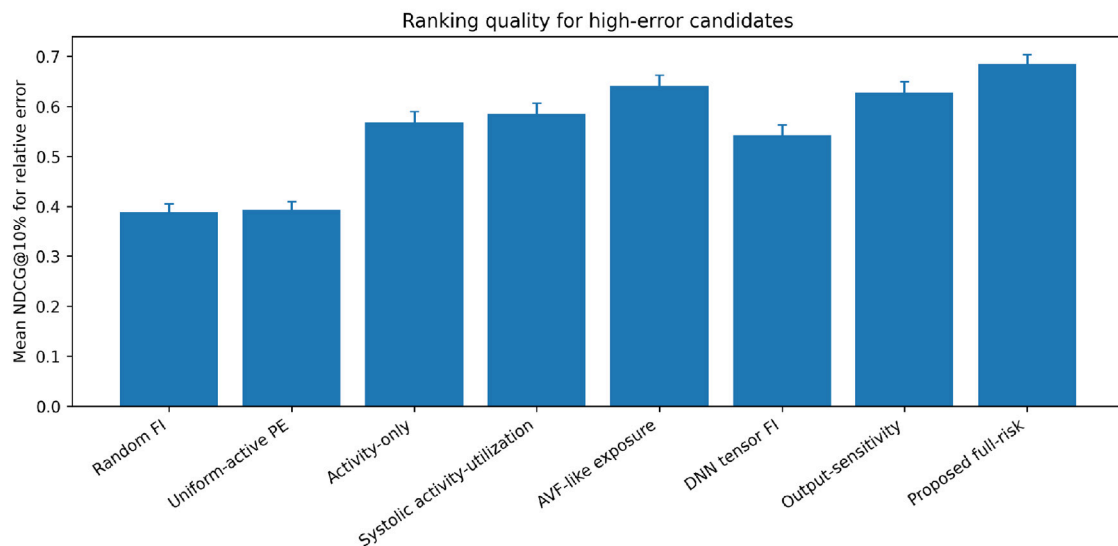


Figure 5. NDCG@10% for ranking high-error candidates.

4.4. Workload and Array-Size Sensitivity

The workload-shape results test whether the ranking result is driven only by the hotspot-sparse stress case. The proposed full-risk method remains competitive across dense, transformer-like, convolution-im2col, and hotspot-sparse shapes. The array-size results show that the prioritization benefit persists across 8×8 , 16×16 , and 32×32 arrays. Figures 6 and 7 report the workload-shape and array-size comparisons, respectively.

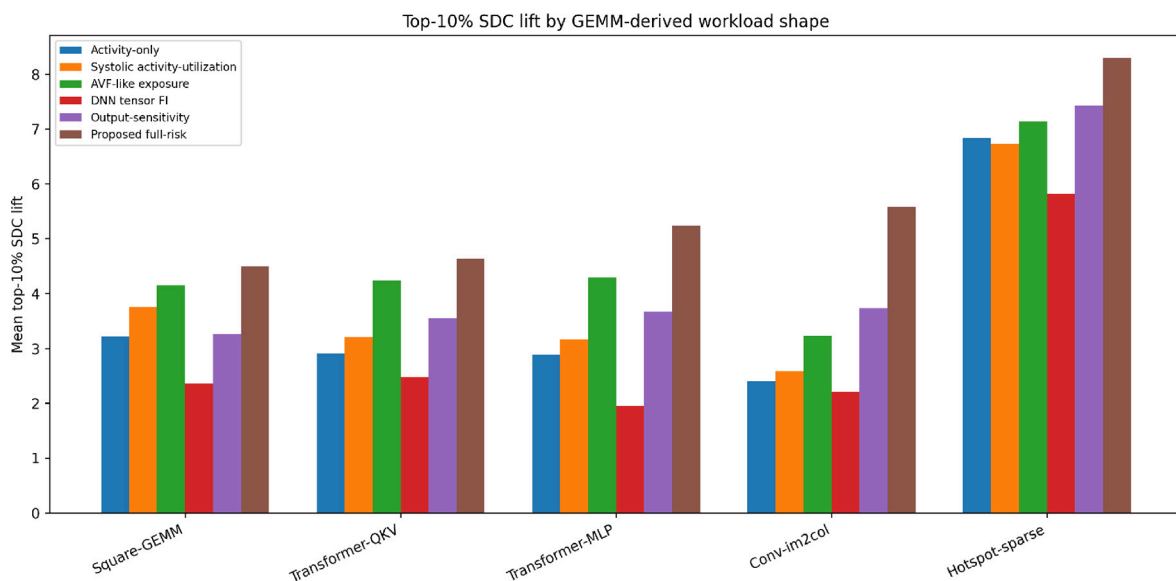


Figure 6. Top-10% threshold-exceedance lift by GEMM-derived workload shape.

Table 7. Paired configuration-level comparison of top-10% SDC lift.

Baseline Compared with Proposed	Mean Paired Lift Difference	Wins/Configs	Losses	Wilcoxon p
Activity-only	2.00	27/30	0	2.78×10^{-6}
Systolic activity-utilization	1.76	27/30	0	2.8×10^{-6}
AVF-like exposure	1.04	19/30	3	0.00024
DNN tensor FI	2.69	28/30	0	1.88×10^{-6}
Output sensitivity	1.32	22/30	4	0.00014

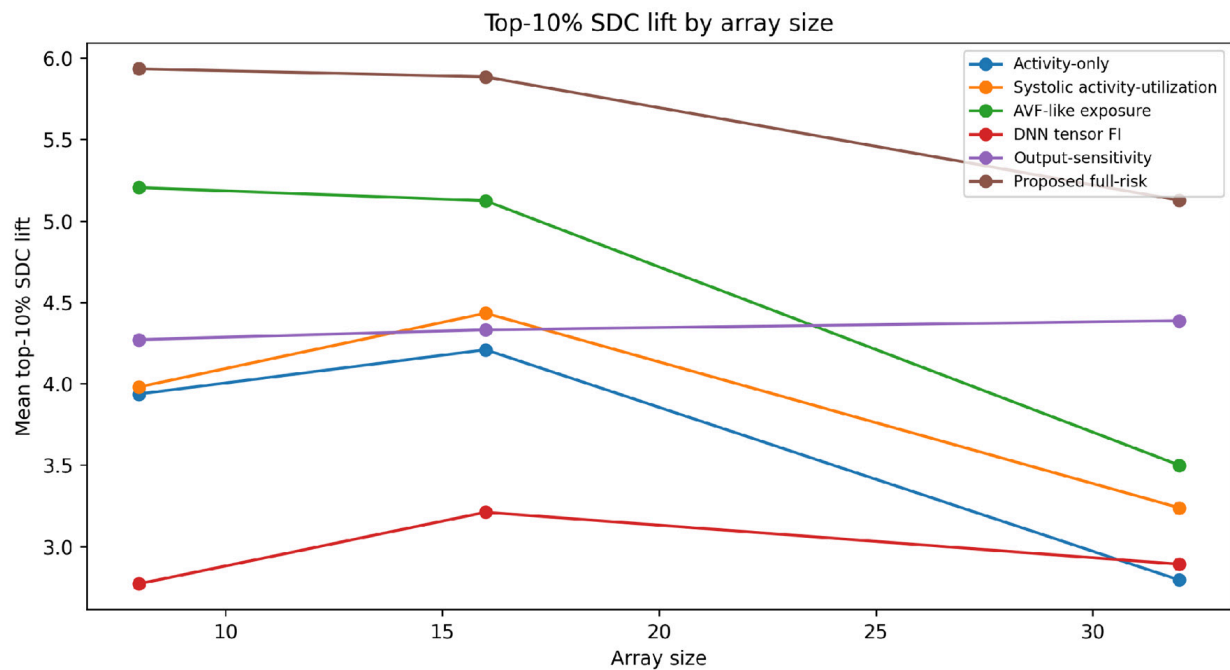


Figure 7. Top-10% threshold-exceedance lift by array size.

4.5. Threshold Sensitivity

The full-risk score remains strongest across all tested thresholds in this simulation protocol. This reduces the risk that the primary ordering is an artifact of the selected $\tau = 0.020$ operating point, while not converting τ into a universal application-level accuracy specification. Table 8 reports the threshold-specific lift values, and Figure 8 shows the corresponding trend.

Table 8. Threshold sensitivity of top-10% SDC-proxy lift.

SDC Threshold	Base Threshold-Exceedance Fraction	Systolic Activity	AVF-Like	Output-Sensitivity	Proposed
0.016	0.052	3.39×	3.87×	3.74×	4.67×
0.020	0.018	3.88×	4.61×	4.33×	5.65×
0.024	0.008	3.95×	5.12×	4.82×	6.60×
0.028	0.003	3.57×	4.94×	6.21×	7.20×

4.6. Outcome-Model Sensitivity and Outcome-Dependence Analysis

The primary balanced outcome model and the proposed ranking score use overlapping high-level feature families. This creates outcome-model dependence that cannot be removed within a synthetic study. The following analysis therefore asks a deliberately narrower question: whether the ordering of methods changes when the hidden outcome construction is altered while the candidate pool and ranking scores are held fixed. The variants emphasize activity, stress, tensor sensitivity, path class, or mixed random coefficients. For each variant, the positive class is defined by a per-configuration high-error tail so that the task remains a rare-event ranking problem.

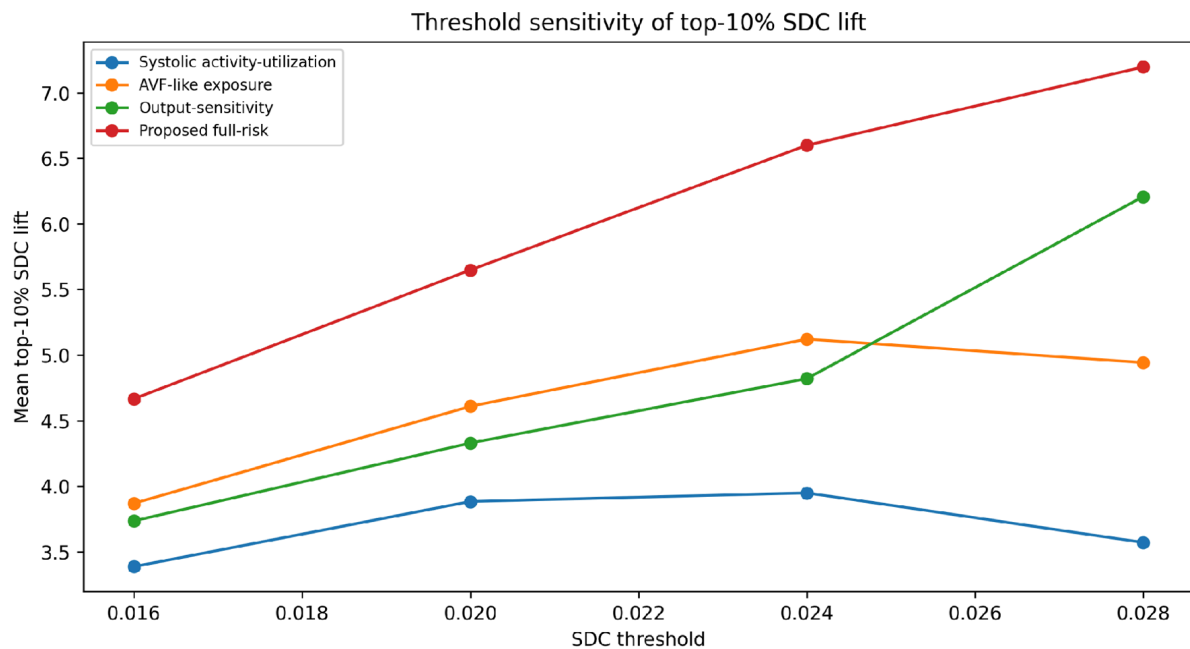


Figure 8. Threshold sensitivity of top-10% threshold-exceedance lift.

This sensitivity analysis is not independent hardware validation and does not establish a silicon-level failure model. It tests whether the proposed ranking advantage disappears under altered synthetic outcome assumptions. The proposed score is not always the best method: in the activity-dominant variant, AVF-like exposure slightly outperforms it, and in the tensor-dominant variant, output-sensitivity becomes the strongest method. These reversals are expected and informative. They show that the proposed score is not guaranteed to win by construction; its relative advantage is associated with multifactorial synthetic outcomes that depend jointly on activity, composite stress, tensor sensitivity, and path class.

The values in Table 9 are computed under the outcome-model sensitivity protocol, which defines SDC using a per-configuration high-error tail. They are therefore not expected to match exactly the primary threshold-based values reported in Table 6.

Table 9. Outcome-model sensitivity of top-10% threshold-exceedance lift.

Hidden Outcome Variant	Proposed Top-10 Threshold-Exceedance Lift	Best Competing Method	Best Competitor Lift	Proposed Rank
Original balanced	5.57×	AVF-like exposure	4.45×	#1
Activity-dominant	5.45×	AVF-like exposure	5.52×	#2
Stress-dominant	4.90×	Output-sensitivity	4.00×	#1
Tensor-dominant	4.76×	Output-sensitivity	6.12×	#2
Path-dominant	4.48×	Output-sensitivity	4.17×	#1
Random-coefficient mix	5.33×	AVF-like exposure	4.31×	#1

Notes: #1 and #2 denote first and second rank, respectively, among the evaluated methods.

5. Discussion

5.1. Interpretation of the Main Result

The main result is that the proposed full-risk score improves fixed-pool prioritization over simple ablations and principle-based baselines in the proposed architecture-level simulation environment. The strongest competitors are AVF-like exposure and output-sensitivity, representing architectural exposure and functional output effect. The full-risk

score combines activity, composite spatial stress, tensor sensitivity, and path class in a single candidate-level ordering rule.

5.2. Importance of the Fixed-Pool Design

The fixed-pool design prevents a common flaw in risk-guided fault-injection studies. If each method sampled different candidates or used different perturbation magnitudes, improved SDC discovery could be attributed to sampling bias or stronger injected faults. In this study, all methods rank exactly the same candidate pool. Fault magnitude and relative output error are computed independently of the ranking method. The comparison therefore evaluates ranking quality under a shared validation budget.

5.3. Why the Proposed Method Underperforms in Tensor-Dominant Variants

The tensor-dominant outcome variant is a useful stress test for the full-risk score. When the hidden outcome model is dominated primarily by tensor sensitivity, the output-sensitivity baseline outperforms the full-risk score. This is expected because output-sensitivity is specialized for that assumption, whereas the full-risk score distributes priority across activity, composite spatial stress, tensor sensitivity, and path class. This result clarifies the intended domain: the full-risk score is most appropriate when the synthetic risk mechanism is multifactorial rather than dominated by a single tensor-level effect.

5.4. Integration into Practical Validation Workflows

In a practical validation workflow, the proposed score would be used before expensive dynamic fault injection. Workload and dataflow descriptors could be obtained from an architecture simulator or mapping tool and converted into activity, residency, relay, composite-stress, and tensor-sensitivity maps. The ranking step would then identify a reduced set of high-priority candidate classes for downstream evaluation in more detailed simulations, emulations, RTL fault injections, FPGA prototyping, or silicon validations. The method is therefore a front-end prioritization layer; it does not replace existing simulators, EDA tools, or implementation-level validation.

Table 10 makes the intended handoff from architecture-level candidates to future RTL-level experiments explicit. It is a mapping aid, not an RTL validation result, and it identifies signal groups that could be targeted in a downstream fault-injection campaign.

Table 10. Architecture-level path classes and possible downstream RTL validation hooks.

Architecture-Level Class	Functional Meaning	Possible Downstream RTL Validation Hook
MAC	Local multiply/product contribution	Multiplier output, product register, or MAC input bus
ACC	Partial-sum update and feedback	Accumulator register, adder output, or partial-sum feedback path
FWD	Inter-PE operand propagation	Activation/weight forwarding register or inter-PE link

This mapping is included to clarify the scope and the future validation path. The present study does not perform RTL, FPGA, or silicon fault injection, but the path classes are formulated so that they can be translated to concrete RTL signal groups in a downstream study.

5.5. Mathematical and Computational Reproducibility

The revised formulation defines the candidate tuple, fixed-pool protocol, feature maps, perturbation-magnitude distribution, synthetic relative-output-error proxy, threshold, ranking scores, and statistical tests explicitly. The computational overhead is linear in the number of candidates plus a sorting or top-k selection step. Since the score maps are computed once per configuration, the method is intended to reduce downstream dynamic fault-injection effort rather than add another expensive validation stage. This reproducibility applies to the controlled screening study; it does not by itself provide physical calibration.

6. Limitations and Scope Boundaries

First, this study targets architecture-level prioritization. It does not replace RTL simulation, gate-level fault injection, physical thermal modeling, FPGA validation, or silicon measurement. The candidate faults are transient functional perturbations rather than calibrated bit flips, device-level defects, or cycle-accurate events. Table 10 provides only a prospective mapping from path classes to RTL signal groups and should not be read as RTL validation.

Second, the outcome model is synthetic. The feature maps and the balanced outcome model share broad abstract descriptors, and the composite stress map is intentionally derived from activity, residency, and relay burden. Outcome-model sensitivity reveals that rankings can reverse under altered assumptions, but it does not substitute for independent RTL, FPGA, or silicon validation. The reported results are controlled evidence about ordering behavior within the defined screening environment, not empirical proof of silicon-level failure rates.

Third, the baselines are principle-based rankings inspired by established methodological families. They are useful for controlled comparison, but they are not complete implementations of SCALE-Sim, MAESTRO, TensorFI, PyTorchFI, or commercial EDA workflows.

Fourth, the workload set consists of GEMM-derived shapes rather than complete end-to-end neural-network inference traces. The shapes cover dense, transformer-like, convolution-im2col-like, and stress-test sparse cases, but future work should evaluate full application traces.

Fifth, the thresholded relative-output-error criterion is an operational SDC proxy. It should not be interpreted as a universal application-level accuracy requirement or a model of an explicit hardware detection mechanism. Application-specific outcome criteria would be needed for deployment studies.

7. Future Work

Future work should integrate the ranking method with full systolic-array simulators and real DNN inference traces; validate the path-class handoff with RTL or FPGA fault injection; calibrate stress and timing features against RTL, gate-level, or silicon telemetry; and extend the outcome model to bit-level and permanent-fault mechanisms. A particularly important next step is evaluation against an independent implementation-level outcome model rather than a synthetic proxy that shares feature families with the ranker.

8. Conclusions

This paper presented an architecture-level risk-guided fault-injection prioritization method for systolic AI accelerators and evaluated it against principle-based policies on a fixed candidate pool. Across 30 configurations and 21,000 candidate faults, the full-risk score achieved the strongest top-10% threshold-exceedance lift, AUPRC, NDCG@10%, and

rank correlation with relative output error in the proposed simulation environment. It outperformed activity-only, systolic activity-utilization, AVF-like exposure, DNN tensor FI, and output-sensitivity policies under a protocol where every policy ranked the same candidates and fault magnitude was independent of the score. Paired tests, threshold sensitivity, and outcome-model sensitivity characterize the result while delimiting its scope. The method is best interpreted as an architecture-level screening tool for prioritizing expensive validation effort, not as a replacement for implementation-level reliability signoff.

Funding: This research received no external funding.

Data Availability Statement: The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Kung, H.T. Why systolic architectures? *Computer* **1982**, *15*, 37–46. [\[CrossRef\]](#)
2. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. In-Datcenter Performance Analysis of a Tensor Processing Unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, ON, Canada, 24–28 June 2017; pp. 1–12. [\[CrossRef\]](#)
3. Chen, Y.-H.; Emer, J.; Sze, V. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Republic of Korea, 18–22 June 2016; pp. 367–379. [\[CrossRef\]](#)
4. Sze, V.; Chen, Y.-H.; Yang, T.-J.; Emer, J.S. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* **2017**, *105*, 2295–2329. [\[CrossRef\]](#)
5. Chen, T.; Du, Z.; Sun, N.; Wang, J.; Wu, C.; Chen, Y.; Temam, O. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Salt Lake City, UT, USA, 1–5 March 2014; pp. 269–284. [\[CrossRef\]](#)
6. Chen, Y.; Luo, T.; Liu, S.; Zhang, S.; He, L.; Wang, J.; Li, L.; Chen, T.; Xu, Z.; Sun, N.; et al. DaDianNao: A Machine-Learning Supercomputer. In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Cambridge, UK, 13–17 December 2014; pp. 609–622. [\[CrossRef\]](#)
7. Samajdar, A.; Zhu, Y.; Whatmough, P.; Mattina, M.; Krishna, T. SCALE-Sim: Systolic CNN Accelerator Simulator. *arXiv* **2018**, arXiv:1811.02883.
8. Kwon, H.; Chatarasi, P.; Pellauer, M.; Parashar, A.; Sarkar, V.; Krishna, T. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflows: A Data-Centric Approach Using MAESTRO. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Columbus, OH, USA, 12–16 October 2019; pp. 754–768. [\[CrossRef\]](#)
9. Mukherjee, S.S.; Weaver, C.; Emer, J.; Reinhardt, S.K.; Austin, T. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor. In Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), San Diego, CA, USA, 3–5 December 2003; pp. 29–40. [\[CrossRef\]](#)
10. Wang, N.J.; Quek, J.; Rafacz, T.M.; Patel, S.J. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. In Proceedings of the International Conference on Dependable Systems and Networks (DSN), Florence, Italy, 28 June–1 July 2004; pp. 61–70.
11. Reis, G.A.; Chang, J.; Vachharajani, N.; Rangan, R.; August, D.I. SWIFT: Software Implemented Fault Tolerance. In Proceedings of the International Symposium on Code Generation and Optimization (CGO), San Jose, CA, USA, 20–23 March 2005; pp. 243–254. [\[CrossRef\]](#)
12. Cho, H.; Mirkhani, S.; Cher, C.-Y.; Abraham, J.A.; Mitra, S. Quantitative Evaluation of Soft Error Injection Techniques for Robust System Design. In Proceedings of the 50th Annual Design Automation Conference (DAC), Austin, TX, USA, 29 May–7 June 2013.
13. Chen, Z.; Narayanan, N.; Fang, B.; Li, G.; Pattabiraman, K.; DeBardeleben, N. TensorFI: A Flexible Fault Injection Framework for TensorFlow Applications. *arXiv* **2020**, arXiv:2004.01743.
14. Mahmoud, A.; Aggarwal, N.; Nobbe, M.; Vicarte, J.R.S.; Adve, S.V.; Fletcher, C.W.; Frosio, I.; Hari, S.K.S. PyTorchFI: A Runtime Perturbation Tool for DNNs. In Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Valencia, Spain, 29 June–2 July 2020.
15. Taheri, M.; Ahmadilivani, M.H.; Jenihhin, M.; Daneshtalab, M.; Raik, J. APPRAISER: DNN Fault Resilience Analysis Employing Approximation Errors. *arXiv* **2023**, arXiv:2305.19733.

16. Li, G.; Hari, S.K.S.; Sullivan, M.; Tsai, T.; Pattabiraman, K.; Emer, J.; Keckler, S.W. Understanding Error Propagation in Deep Learning Neural Network Accelerators and Applications. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver, CO, USA, 12–17 November 2017. [\[CrossRef\]](#)
17. Vatajelu, E.I.; Di Natale, G.; Prinetto, P. Reliability of Deep Neural Networks. In Proceedings of the IEEE VLSI Test Symposium (VTS), Monterey, CA, USA, 23–25 April 2019.
18. Hegde, K.; Agrawal, R.; Yao, Y.; Fletcher, C.W. Morph: Flexible Acceleration for 3D CNN-based Video Understanding. In Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Fukuoka, Japan, 20–24 October 2018.
19. Skadron, K.; Abdelzaher, T.; Stan, M.R. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. In Proceedings of the 8th International Symposium on High-Performance Computer Architecture (HPCA), Cambridge, MA, USA, 2–6 February 2002. [\[CrossRef\]](#)
20. Huang, W.; Ghosh, S.; Velusamy, S.; Sankaranarayanan, K.; Skadron, K.; Stan, M.R. HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2006**, *14*, 501–513. [\[CrossRef\]](#)
21. Borkar, S. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro* **2005**, *25*, 10–16. [\[CrossRef\]](#)
22. Tiwari, A.; Torrellas, J. Facelift: Hiding and Slowing Down Aging in Multicores. In Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Lake Como, Italy, 8–12 November 2008; pp. 129–140. [\[CrossRef\]](#)
23. Micikevicius, P.; Stosic, D.; Burgess, N.; Cornea, M.; Dubey, P.; Grisenthwaite, R.; Ha, S.; Heinecke, A.; Judd, P.; Kamalu, J.; et al. FP8 Formats for Deep Learning. *arXiv* **2022**, arXiv:2209.05433.
24. Noune, B.; Jones, P.; Justus, D.; Masters, D.; Luschi, C. 8-bit Numerical Formats for Deep Neural Networks. *arXiv* **2022**, arXiv:2206.02915.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.