

# Late Breaking Results: Never-Stopping Inference: Self-Healing AI Accelerators on SRAM-FPGAs

Eleonora Vacca, Giorgio Cora, Luca Sterpone  
 Dipartimento di Automatica e Informatica (DAUIN)  
 Politecnico di Torino, Italy  
 {eleonora.vacca,giorgio.cora,luca.sterpone}@polito.it

**Abstract**—This work presents a self-healing runtime for AI accelerators on SRAM-based FPGAs that combines online fault detection with fine-grained partial reconfiguration to ensure continuous inference execution. The framework dynamically isolates and repairs faulty regions while remapping workloads to healthy resources, eliminating the need for redundant hardware and system downtime. The proposed approach reduces recovery latency by 3 orders of magnitude compared to the state-of-the-art.

**Index Terms**—AI, Systolic Arrays, Dynamic Partial Reconfiguration, Fault Detection, Fault Correction, Reliability

## I. INTRODUCTION

SRAM-based FPGAs are widely used for AI accelerators due to their configurability and energy-efficient parallelism, but they suffer from a core reliability issue: the configuration memory (CRAM). Because CRAM encodes the effective netlist, LUT truth tables, placement, and routing, radiation-induced bitflips can alter logic or interconnect at runtime, silently corrupting computation or even stalling the design until repaired [1]. This behavior is qualitatively different from transient register or BRAM faults, as a single CRAM upset can persistently rewire the circuit.

Traditional mitigation relies on redundancy, such as Triple Modular Redundancy (TMR) or selective replication, but these approaches are often impractical for large FPGA-based AI accelerators [2] - [5]. Full TMR is typically too resource-intensive, and scaling down the design to fit TMR negates the performance benefits. Selective redundancy lacks generality, since vulnerability varies across models, datasets, and quantization schemes, and identifying sensitive components requires extensive and non-transferable fault-injection analysis.

Memory scrubbing (e.g., AMD’s SEM IP) is another standard technique, but it is too slow for inference workloads: scrubbing operates on millisecond timescales while accelerator pipelines run at nanosecond cycles, allowing errors to propagate before correction. ECC also cannot detect all CRAM corruption patterns, and the scrubber itself, implemented in programmable logic, remains vulnerable. If the SEM controller faults while holding configuration access, it can deadlock the device and force a full reboot, causing seconds-long downtime.

To address these limitations, recent work explores dynamic partial reconfiguration (DPR) [6] [7] as a fault-tolerance primitive, repairing only corrupted CRAM regions without rebooting [8] [9]. This paper advances that direction by introducing a fine-grained, self-healing runtime for AI accelerators on SRAM-based FPGAs. The system combines lightweight online fault detection through state-of-the-art checksum based computation [10]- [13], localized DPR, and adaptive workload remodulation to preserve continuous inference. We prototype the approach

on systolic-array (SA) accelerators using a checksum-based detection scheme capable of locating faults to a single SA column. This allows column-level partial reconfiguration, yielding roughly three orders of magnitude lower recovery latency compared to full-array reconfiguration. Crucially, the method maintains uninterrupted inference and eliminates system downtime.

## II. IMPLEMENTATION

Our proposed methodology integrates a fault-detection and correction mechanism specifically tailored for SAs implemented on SRAM-based FPGAs. First, we augment the SA with a checksum-based technique capable of detecting errors in the matrix multiplications performed by the core. Through lightweight modifications to the accelerator, when the testing mode is activated, dedicated test vectors are appended to the main computation. As these vectors propagate through the PE grid, they produce  $N$  checksums corresponding to the weight data stored in each of the  $N$  columns. These checksums are then compared against reference values computed in parallel by an external bank of accumulators, which receives the same weight data. The result of this comparison generates an error vector that identifies which SA columns are faulty.

The SA pipeline has been modified such that, when testing mode is enabled, it pauses the injection of new operations into the pipeline until the checksum comparison is completed [11]. If a fault is detected, the hardware automatically excludes all faulty columns from the computation and places them under recovery. During the recovery phase, each affected column undergoes partial reconfiguration. Meanwhile, the workload is dynamically redistributed across the healthy resources, and any operations that cannot be executed due to missing columns are properly logged so that they can be executed after all currently queued operations have completed. A practical example of this mechanism is illustrated in Fig. 1, which demonstrates how computation is managed when, for simplicity, only one column is faulty and a convolution is performed on an RGB image. To execute the convolution on the SA, the operation is reformulated as a general matrix multiplication. The RGB image undergoes an  $\text{Img2Col}$  transformation, and the filters are vectorized. To simplify recovery and allow hardware-level reorganization, computations are structured by channel. This means that, for each filter, the kernels corresponding to the same channel are grouped within the same weight matrix, since they will be multiplied by the same input-channel matrix. The partial results obtained for each channel are then accumulated to produce the final output feature map for each filter. Figure 1b illustrates the case of a  $3 \times 3$  SA and a convolution using 6

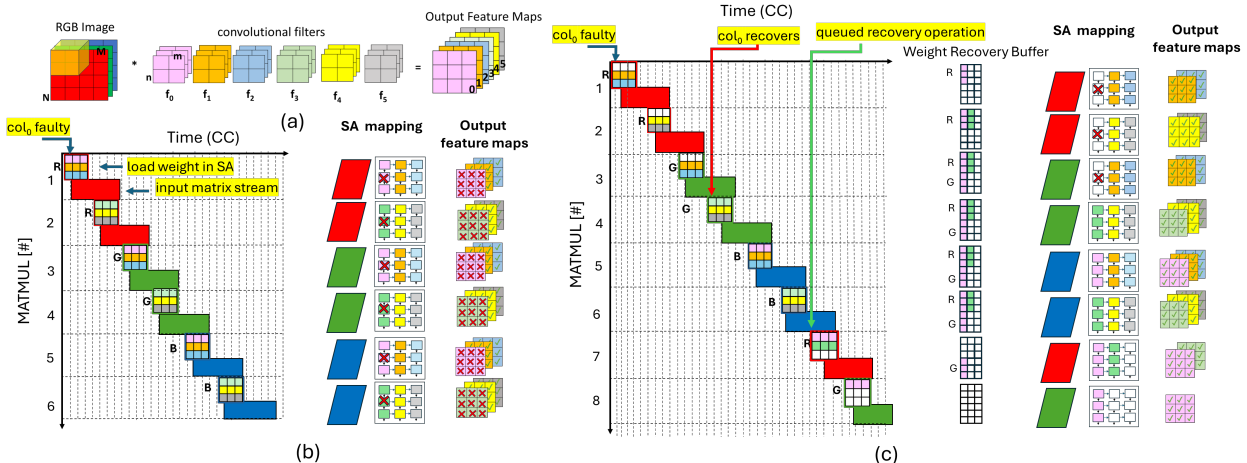


Fig. 1. Recovery flow when a fault occurs in column 0 of the systolic array (SA). (a) The convolution operation used as an example workload. (b) The standard SA pipeline and data-to-resource mapping, including the diagonalized input matrix, the stationary weight matrix and output feature map for each matrix multiplication, when no recovery is applied. (c) The modified execution flow when the recovery routine is enabled.

filters. With a  $3 \times 3$  array, each channel’s computation is split into two consecutive and independent matrix multiplications, each requiring loading the corresponding weight matrix from memory and multiplying it by the appropriate input matrix. Thus, each operation begins with a load-weights phase, where the weights are fetched from the weight buffer and placed into the SA grid. Once the first PE row is fully loaded, the input matrix can be streamed in. In the example, we assume that at  $t = 0$ , testing mode identifies column 0 as faulty. If no corrective actions were taken and data continued to map onto the faulty column, the computations for the R, G, and B channels of filters  $f_0$  and  $f_3$  would be incorrect. Figure 1(c) shows how our method resolves the issue. As soon as column 0’s error flag is raised, the column is disabled and its associated weights are temporarily stored in a recovery buffer. Because operations 1 and 2 share the same input matrix (the R channel), the corresponding weights that cannot be mapped due to the faulty column are stored in adjacent memory locations in the buffer, facilitating their retrieval during recovery. During operation 3, the column remains faulty; however, the computation now concerns the G channel, so the  $f_0$  G-channel weights are stored in a different memory region. After partial reconfiguration, column 0 becomes operational again, and the workload proceeds normally through operation 6, completing the blue-channel computations. Before releasing the unit, we automatically feed the pipeline with recovery operations 7 and 8 to process the previously skipped computations: the missing R-channel results for filters  $f_0$  and  $f_3$ , and the missing G-channel result for filter  $f_0$ , respectively. This method provides two key advantages. First, compared to approaches that rely on a single spare column—which can handle only one faulty column at a time—our technique offers greater flexibility. Because reconfiguration and workload distribution are dynamically adapted to the set of healthy columns at each operation, the system can manage an arbitrary number of faulty columns. Note that partial reconfiguration is performed one column at a time; as each column is restored, it immediately re-enters the computation at the next available operation. Consequently, the effective capacity of the SA recovers progressively, column by column.

#MAC per column	Reconfiguration Time (ms)			
	1 column	2 columns	4 columns	NxN SA
6	0.59	1.18	2.36	5.86
10	0.65	1.30	2.60	9.67
14	0.79	1.58	3.16	13.83
18	1.01	2.02	4.04	18.01
22	1.19	2.38	4.76	21.79

Resource Type	Plain	Full Recon.	Column-wise Reconf.
LUT	3627	4564	5501
FF	4336	7578	6892
BRAM	139	175	179
DSP	218	210	210

Second, the use of fine-grained reconfiguration allows inference to continue uninterrupted. Unlike full-accelerator reconfiguration, which incurs system downtime, our approach maintains continuous operation throughout the repair process.

### III. RESULTS

We evaluated reconfiguration latency against the baseline design that reconfigures the entire SA, implementing the design on ZCU102 SoC, embedding finFET Ultrascale FPGA. Results in Table 1 show that with fine-grained column-level partial reconfiguration, the reconfiguration time drops by several milliseconds, especially for larger columns. For a 22-MAC column, it reduces from 21 ms to  $\sim 1.2$  ms ( $\sim 17\times$ ). For multiple faulty columns, total repair time is simply the sum of the per-column reconfiguration latencies, yet it still remains far below the cost of full-array reconfiguration. Unlike a full reconfiguration—where the accelerator is entirely offline and must later re-execute the corrupted work—our fine-grained approach avoids downtime altogether. Recovery operations are automatically queued and absorbed by the pipeline, reducing the effective recovery overhead from milliseconds to microseconds.

### IV. CONCLUSIONS

We propose a fine-grained, column-level partial reconfiguration approach for systolic arrays. When a fault occurs, only the affected columns are reconfigured while others continue processing, drastically reducing the recovery latency and eliminating system downtime.

## REFERENCES

- [1] Heather Quinn, "Radiation effects in reconfigurable FPGAs", 2017 *Semicond. Sci. Technol.*, <https://doi.org/10.1088/1361-6641/aa57f6>
- [2] N. Cherezova et al. "FORTALESA: Fault-tolerant reconfigurable systolic array for DNN inference", *Microprocessors and Microsystems*, Volume 119, 2025, <https://doi.org/10.1016/j.micpro.2025.105222>.
- [3] Y. Zhao et al. "FSA: An Efficient Fault-tolerant Systolic Array-based DNN Accelerator Architecture," 2022 IEEE 40th International Conference on Computer Design (ICCD), Olympic Valley, CA, USA, 2022, pp. 545-552, doi: 10.1109/ICCD56317.2022.00086.
- [4] H. Lee et al. "An Area-Efficient Systolic Array Redundancy Architecture for Reliable AI Accelerator," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 10, pp. 1950-1954, Oct. 2024, doi: 10.1109/TVLSI.2024.3421563.
- [5] A. Ruospo et al. "Selective Hardening of Critical Neurons in Deep Neural Networks," 2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), Prague, Czech Republic, 2022, pp. 136-141, doi: 10.1109/DDECS54261.2022.9770168.
- [6] W. Lie and W. Feng-yan, "Dynamic Partial Reconfiguration in FPGAs," 2009 Third International Symposium on Intelligent Information Technology Application, Nanchang, China, 2009, pp. 445-448, doi: 10.1109/IITA.2009.334.
- [7] Yufan Lu et al., "FPGA based Adaptive Hardware Acceleration for Multiple Deep Learning Tasks", 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoc), pp.204-209, 2021.
- [8] Matteo Monopoli et al., "Exploring Key Aspects of Soft GPGPU Computing for On-board Acceleration of Artificial Intelligence Algorithms in Space Applications", 2023 European Data Handling & Data Processing Conference (EDHPC), pp.1-6, 2023.
- [9] G. Cora et al., "RePAIR: Reconfigurable Platform for AI Resilience Within RISC-V Ecosystem", *Applied Reconfigurable Computing. Architectures, Tools, and Applications (ARC) 2025*, doi:/10.1007/978-3-031-87995-1\_5
- [10] L. Chen, et al., "Extending checksum-based ABFT to tolerate soft errors online in iterative methods," 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2014, pp. 344-351.
- [11] E. Vacca et al. "RunSAFER: A Novel Runtime Fault Detection Approach for Systolic Array Accelerators", IEEE 41st International Conference on Computer Design (ICCD), 2023. doi:10.1109/ICCD58817.2023.00095
- [12] F. Libano et al., "Efficient Error Detection for Matrix Multiplication With Systolic Arrays on FPGAs," in *IEEE Transactions on Computers*, vol. 72, no. 8, pp. 2390-2403A.
- [13] E. Aliagha et al., "SCISSORS: System Level Error Detection for Enabling Near-Threshold Operating Systolic Arrays," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, doi: 10.1109/TCAD.2025.3584733.