Scalable Reliability Assessment of DNNs through Simultaneous Fault Injection

Rafael Billig Tonetto, Marcello Traiola, Fernando Fernandes, Angeliki Kritikakou

This work has been accepted at IEEE/ACM Design Automation Conference (DAC) 2026

Scalable Reliability Assessment of DNNs through Simultaneous Fault Injection

Rafael Billig Tonetto, Marcello Traiola, Fernando Fernandes dos Santos, Angeliki Kritikakou
Univ Rennes, CNRS, Inria, IRISA, UMR 6074, Rennes, France
{rafael.billig-tonetto,marcello.traiola,fernando.fernandes-dos-santos,angeliki.kritikakou}@inria.fr

Abstract

Deep Neural Networks (DNNs) are increasingly deployed in safety-critical applications, where resilience to transient faults is essential. Traditional fault injection methods struggle to scale to large models, while most acceleration techniques depend on specific hardware or architectures. Simultaneous fault injection—injecting multiple faults in a single run—has shown promise, but its applicability to DNNs remains unexplored. In this work, we show that simultaneous faults in DNNs can interact due to their dense connectivity, creating artificial masking effects that lead to false negatives and compromise reliability assessment. To address this, we propose a method to mitigate fault interactions during simultaneous injection, ensuring accurate results. We further accelerate the process by pruning non-critical inputs from the batch, achieving additional speedup. To our knowledge, this is the first approach enabling accurate and efficient simultaneous fault injection in DNNs across abstraction levels. We evaluate nearly 42 million injections at both software (SW) and RTL, achieving very low false negatives (down to 0%, avg. 0.2%) and speedups of 3.82× (RTL) and 5.29× (SW).

1 Introduction

DNNs have become pervasive across domains such as computer vision, data science, and natural language processing. Gains in DNN accuracy are largely achieved by increasing their depth and parameter counts, at the cost of higher computational demands. At the same time, the need for efficient reliability assessment methods has grown, as modern technologies are increasingly susceptible to transient faults [7, 8]. This issue is particularly important for DNNs applicable to critical domains in which random bit-flips can compromise system reliability [13, 21].

Fault injection is a fundamental methodology for reliability assessment, and realistic approaches must account for fault-masking effects across different abstraction levels [20, 25]. Therefore, low-level fault injection (e.g., at RTL level) provides more realistic assessments, but substantially increases the time and cost of the assessment. Coupled with the growing complexity of DNNs, this creates challenging trade-offs between reliability assessment accuracy and experimentation time, highlighting the need for scalable fault injection approaches.

The majority of conventional approaches inject a single fault during one workload execution. To speed up the reliability assessment, existing approaches explore different layers of the system stack. Although such approaches are suitable for rather small models, e.g., MLPs [16, 34, 35], LeNet-5 [6, 16, 29, 32], AlexNet [29, 30, 34, 35], and ResNet-18/20 on small datasets [30, 32], reliability assessment of highly complex DNN models requires much more efficient solutions in order to reduce the need of large-scale and very expensive cluster infrastructures, such as those used by large companies [8, 9, 15, 31]. To overcome this issue, alternative approaches rely on the extraction of RTL error models [3, 14, 29]. Although efficient, these models are often closely tied to the specific hardware architectures, restricting their portability to other architectures. Another strategy for speeding up is to inject faults simultaneously. Simultaneous injection proved effective for general-purpose applications, relying on the fundamental assumption that *the probability of interaction between several randomly injected faults is practically zero [10]*. DNNs are generally considered intrinsically fault-tolerant, which means that the majority of faults do not impact their inference results (i.e., they are *non-critical*) [2]. This intrinsic property suggests the potential to use a simultaneous injection to enhance the reliability assessment of DNNs, as a significant portion of the faults will be masked even if injected simultaneously. However, DNNs have a densely connected structure, which threatens the fundamental assumption of no interaction among simultaneously injections.

To explore simultaneous fault injection in DNNs, we conduct an empirical analysis revealing that DNN workloads are highly prone to fault interactions. Due to these interactions, masking effects may occur, which lead to the misclassification of the injected faults as non-critical (called *false negatives*, more details in Section 3.4). As a result, the rate of faults categorized as critical is significantly lower than conventional approaches, which inject a single fault at a time (from now on referred to as *fault recall*). This ultimately compromises the accuracy of the reliability assessment, making simultaneous fault injection ineffective for DNNs, when applied without an appropriate methodology. We address this limitation by proposing a novel methodology that enables simultaneous fault injection within a single DNN forward pass. To our knowledge, no prior work has proposed an accelerated reliability assessment for DNNs based on simultaneous injection of faults applicable at different abstraction levels and DNN models. More precisely, we propose (i) a methodology to mitigate false negatives occurring due to fault interactions, achieving accurate reliability assessment, and (ii) an acceleration strategy, that early prunes high-confidence inputs from the input batch, to reduce the total amount of needed DNN forward passes. The proposed methodology is platform-independent and, according to our results, the required experimental time scales sub-linearly with DNN size, largely due to the high number of non-critical faults observed in larger networks. This shows that *simultaneous fault injection is a promising path to efficiently scale the reliability assessment of DNNs*. To evaluate our approach, we conducted fault injection experiments at both the SW and RTL abstraction levels. The obtained results show that the proposed approach reduces the experimental time on average by 3.82× for RTL and 5.29× for SW, compared to conventional approaches for DNNs, which inject a single fault at a time, while maintaining an average of fault recall equal to 99.78% for RTL and 99.94% for SW. In summary, our main contributions are:

- A methodology that enables simultaneous fault injection for DNNs, taking into account fault interactions, resulting in low fault negatives and accurate assessment of the criticality of faults.

- A strategy to further reduce the required time by early pruning the non-critical high-confidence inputs from the batch during reliability assessment.
- Scalability is demonstrated across growing model sizes through extensive experiments involving the injection of nearly $42M$ faults in various quantized models and at two different abstraction levels. i.e., SW and RTL.

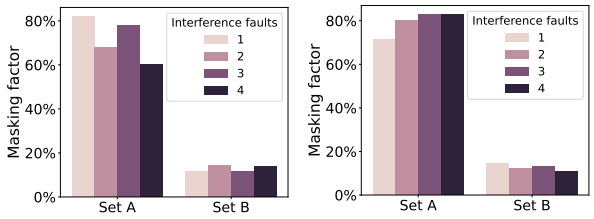
2 Background and Related Work

A large body of research addresses the reliability assessment of DNNs. Existing approaches can be broadly categorized into radiation-based physical injection and simulation-based injection.

Radiation-based approaches [22] have been proposed for the evaluation of GPUs [26] or Tensor Processing Units (TPUs) [23]. Although such approaches provide very accurate reliability analysis, they lack injection controllability and reproducibility, making it difficult to obtain full fault characterization.

Simulation-based injections can be categorized into hardware-level [30], architectural-level targeting user-visible registers [12], and software-level targeting only model parameters [6, 32, 33] or model operators [5]. Software-level methodologies provide fast reliability assessment, but they often yield inaccurate vulnerability results by neglecting fault-masking properties [20] that are only captured by low-level approaches, which consider hardware details. Several simulation-based approaches for DNNS focus on faults in memory [4, 12, 21]. Although such fault models can be exploited without relying on detailed hardware models, memory faults affect only the model parameters (weights and inputs) and do not capture faults occurring during computations. To deal with faults occurring during computations, approaches extract error models from hardware description of the logic [14, 29, 30] or error patterns extraction applicable to GPU kernel operators [3] for fault injection at software layer. These methods reduce the time required for reliability assessment compared to full hardware simulation, but such error models may degrade the reliability assessment accuracy [20] and are typically tightly coupled to the specific hardware architecture. Overall, most existing simulation-based approaches typically inject a single fault at a time. However, as the effect of most faults is usually masked, such conventional approaches have low efficiency, as the majority of injections has no final effect in the DNN execution.

For specific applications, a fault injection technique has been introduced to overcome the inefficiency of conventional injection approaches [10]. The fundamental assumption is that, in specific applications, randomly injected faults rarely interact, so several faults can be injected simultaneously during a single workload execution. Note that the goal is to characterize the effects of single faults and not to evaluate the effect of multiple faults. To manage this process, the faults to be injected during reliability assessment (called *fault list*) are organized in a K-ary tree. Each tree leaf represents a single fault. Internal tree nodes group the faults of their children, and no assumption of fault characteristics are made to form the fault nodes. The reliability assessment starts from the root. When a node is visited, all faults of the node are injected in a single workload run. If no failure occurs (i.e., the workload does not execute correctly), all injected faults are marked as masked and the entire node's subtree is pruned. If a failure does appear, the set of injected faults has to be further explored to identify the responsible fault(s) for the failure. Thus, all node's children have to be visited.



(a) Targeting the layer inputs **(b)** Targeting the layer outputs

Figure 1. Masking probability due to fault interaction on two input sets from ImageNet.

However, the technique proposed in [10] applies only to applications without any fault interactions, which limits its applicability. Most importantly, as shown by our analysis, faults in DNNs are highly prone to interaction due to the densely connected DNN structure, and, in the worst case, critical faults are masked during simultaneous injection, leading to false negatives, highly compromising the accuracy of the reliability assessment.

3 Proposed Reliability Assessment Framework

The section provides an example to motivate our methodology, then presents our proposal to address fault interactions and reduce injection time.

3.1 Motivating Example

To motivate our analysis with an example, we use the first convolution layer of a ResNet18 model. We show the probability of fault masking due to the interaction of faults when injected simultaneously in the a) inputs and b) outputs of the layer. We consider a fault list $\mathcal{S}$ that specifies the faults to be injected during the reliability assessment. From the full fault list $\mathcal{S}$, we extract a sublist $\mathcal{F} \subset \mathcal{S}$ with critical faults, i.e., they change the ResNet18 classification when injected. For each critical fault $f_i \subset \mathcal{F}$, we simultaneously inject from 1 up to 4 *additional* faults (s_i) randomly selected from $\mathcal{S}$ along with f_i and perform a forward pass. Then, we measure the probability for the additional faults s_i to mask f_i (called *masking factor*). Fig. 1 shows the masking factor for two input sets (A and B, 16 inputs each) from ImageNet that have different masking sensitivity levels (for illustration purposes). Firstly, we observe that critical faults ($\mathcal{F}$) have a masking probability higher than zero, which is caused due to the interaction with the simultaneously injected faults. This violates the fundamental assumption made in [10], leading to false negatives that degrade the accuracy of the reliability assessment. Moreover, we observe that the masking sensitivity depends on the inputs used, and some inputs are more sensitive than others, such as set A with respect to set B.

Building upon these insights, our work leverages the approach in [10] to enable simultaneous fault injection applicable for DNNs and at different abstraction levels. Our methodology mitigates the negative impact of fault interactions on the accuracy of the reliability assessment even under scenarios with a high number of faults simultaneously injected, both at SW and RTL levels. The experimental results show that the proposed methodology is fast, accurate, scalable, and independent of the abstraction level.

3.2 Reducing Reliability Assessment Time

During the reliability assessment setup, the list of faults to be injected is organized as a K-ary tree, where each node has at most K children. Alg. 1 summarizes the methodology proposed to visit

the tree-based fault list, and Fig. 2 illustrates it through an example. Our speed-up is attributed to the following two key factors:

1) Simultaneous fault injection: Each visit of a tree node represents a set of faults to inject, simultaneously, in a single model inference. Tree nodes are put in a processing queue, from top to bottom, and are processed in a Breadth-first search fashion, as shown in Alg. 1. A node is considered critical, if, for at least one input in the batch, the injection caused a misclassification, i.e., an incorrect DNN inference. If a node is critical, all its children are pushed to the queue for further evaluation; otherwise, no descendants of the node are evaluated, and thus, evaluation time is saved. Critical faults are identified by traversing the tree and marking all leaf nodes that lead to misclassifications. This aspect is inherited from [10]. Suitable values for the tree's K parameter can be computed based on the expected failure probability of the task, which is input-dependent and assumes the availability of such estimation. We do not make any assumptions on this probability. Rather, we perform design space exploration and verify that a fixed value (K=25) is sufficient for all evaluated classification tasks based on the number of required injections per layer per input to achieve statistical significance.

Since in DNNs the majority of faults is not critical [2], the tree-based approach provides high speedup, as many nodes are not critical, and thus, the entire sub-trees can be pruned all at once. In comparison, conventional approaches (where all faults are injected one at a time – named *sequentially*), directly visit all the leaves of the tree. Therefore, when a substantial number of internal tree nodes is pruned, simultaneous fault injection using the tree-based fault list visits fewer nodes than the sequential injection, thereby reducing the overall time required for reliability assessment.

2) Input pruning: On top, our approach prunes the inputs as the fault injection moves down the tree. In the proposed methodology, inputs are removed from the batch if there is high confidence that they are not sensitive to faults (lines 15-19 in Alg. 1). In DNNs, the input sensitivity to faults is correlated to its classification confidence. To identify such property, we use the confidence gap between the Top-1 and Top-2 confidence scores [18]. A large gap means the model is confident in the classification, making the input generally more robust to faults. In our approach, at a given node, we prune high-confidence inputs from the batch if they are not sensitive, i.e., simultaneous injection did not result in misclassification. We use a τ confidence gap threshold to balance the input pruning ratio against speedup. Lower τ thresholds increase the pruning ratio, and thus, reduce injection time, but also tend to increase fault misclassification (false negatives). In our generic experiments, we empirically observe that using a fixed threshold of $\tau = 12\%$ provides low false negatives and high speedups for all ten models under study. While this is a general threshold that accommodates all tasks, task-specific calibration for this parameter is also possible.

3.3 Dealing with Fault Interaction

We empirically demonstrate in the motivating example that when faults are simultaneously injected in DNN models, they interact, leading to false negatives. This means that critical faults (when injected alone) can be masked and treated as non-critical when injected simultaneously with other faults. Such fault interaction is undesirable, as it degrades the accuracy of the reliability assessment.

We suppress the number of false negatives to a negligible level by using a criticality hint as a proxy to continue moving down the tree, even if a node is labeled non-critical. Most false negatives occur when one or more inputs in the batch exhibit low classification confidence. To address this issue, we avoid pruning non-critical internal tree nodes whenever this condition is met. In other words, even if an internal tree node is considered as non-critical, if at least one input in its batch has low confidence (below the threshold θ as specified in Algorithm 1), we still explore the node's children. By using the confidence gap as a symptom to flag possible critical faults, we can control the sensitivity with which the approach descends the tree by adjusting the θ sensitivity threshold (Alg. 1, line 14). Higher thresholds increase node sensitivity, leading to more visited child nodes and, thus, improved fault coverage, but increased evaluation time. For this work, we used a fixed threshold of $\theta = 5\%$, which we empirically observed to offer a good balance between performance and fault recall across all evaluated models. As future work, we will invest in a calibrating procedure to fine-tune this threshold together with the K parameter that defines the tree structure.

Let us illustrate how the proposed methodology is applied using the example of Fig. 2. Reliability assessment is performed following the tree structure. When a node is visited, all faults are injected simultaneously at each of the non-pruned inputs of the batch. If at least one input fails during injection, i.e. leads to misclassification, the node is considered as critical (colored red). Otherwise, it is non-critical (colored green). The inputs that failed are marked in red. Inputs that did not fail are pruned iff they have high enough confidence; otherwise, they are kept and highlighted in blue. Therefore, a node (and its sub-tree) is pruned when it is non-critical and all inputs are confident. For Fig. 2, the reliability assessment starts from the root node, where the complete input batch *{I1, I2, I3, I4}* is used and all faults *{F1-F6}* are injected simultaneously. The injection

Algorithm 1: Processing of Tree Fault List

Input: Tree T, DNN model M, Batch B
Input: Confidence gap threshold τ, Criticality hint threshold θ
Output: Number of critical faults

```
1  Q ← T.root;
2  while Q ≠ ∅ do
3      node ← Q.pop();
4      critical_node ← False;
5      criticality_hint ← False;
6      foreach i ∈ B do
7          critical_node ← critical_node ∨ inject(M(i), node);
8          δ ← top1_conf(i) − top2_conf(i);
9          criticality_hint ← criticality_hint ∨ (δ < θ);
10     if node is leaf then
11         if critical_node then
12             CriticalFaults ← CriticalFaults + 1;
13     else
14         if critical_node or criticality_hint then
15             foreach i ∈ B do
16                 if input i is correctly classified then
17                     δ ← top1_conf(i) − top2_conf(i);
18                     if δ > τ then
19                         Remove i from B;
20             foreach child ∈ node.children() do
21                 Q.push(child);
22 return CriticalFaults;
```

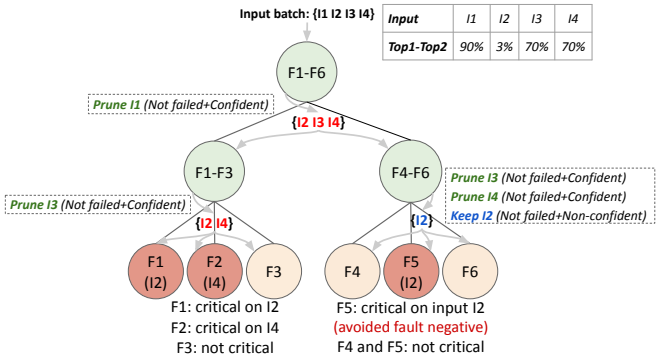


Figure 2. Illustration example: In each tree node, all faults are injected simultaneously and evaluated with the non-pruned inputs. A node is critical if at least one input fails. Non-failing inputs having a high-enough $Top1 - Top2$ confidence gaps are pruned. Critical faults are discovered in the leaf nodes.

causes an incorrect DNN inference for all inputs except *I1*, which has a high confidence gap between Top1 and Top2, and thus, is pruned (i.e., not used in lower nodes). Next, both child nodes are visited using less inputs *{I2, I3, I4}* and faults *{F1-F3}*, and *{F4-F6}*, respectively. The injection of the faults of the first node (*{F1-F3}*) lead to misclassification for *{I2,I4}*, which are propagated to the children, while I3 is pruned as non-critical and confident. Finally, node *{F1}* is critical for the two inputs. The second node is not critical for any input *{I2, I3, I4}* and fault *{F4-F6}*, so it should be normally pruned following existing approaches, such as DAC'16 approach. However, even if *I2* did not fail, its confidence gap is very low (3%). Because the input's low confidence, the absence of a failure may be due to interactions among the injected faults, ultimately leading to a false negatives. Therefore, our methodology does not prune it from the batch, and propagates to the children. This allows us to discover that fault *F5* is actually critical for input *I2*. Without our approach, the node F4-F6 would be pruned, resulting in considering *F5* as masked, contributing to false negatives.

Although injection results vary across DNN models and simulation setups, our methodology provides knobs to easily tune the experiment parameters and efficiently balance the trade-off between accuracy and injection time. For example, for more resilient input sets, one can improve the injection time by choosing smaller K values and fine tuning θ/τ thresholds to the specific task.

3.4 Vulnerability Factors and Approach Accuracy

Our methodology aims at quickly and accurately classifying DNN faults with respect to conventional sequential fault injections. A fault is considered critical if, when injected, the Top-1 label diverges from the fault-free label for at least one input. Otherwise, it is non-critical. The *criticality* of a fault is computed as the percentage of inputs that make the fault critical. For software injections, this is equivalent to the Program Vulnerability Factor (PVF), as it only captures the software-level masking properties [27]. For RTL injections, both software- and hardware-level masking properties are captured, leading to more accurate vulnerability results, measured in terms of the Architectural Vulnerability Factor (AVF) [19].

We use conventional metrics to measure the efficiency of our approach. Correctly classified critical faults are considered true positives (TP), and correctly classified non-critical faults are true negatives (TN). Critical faults that escape detection in the proposed approach are considered false negatives (FN). Note that, by construction, false positives (FP), i.e., incorrectly classifying faults as critical, are not possible in the tree-based approach, as faults reaching the leaf nodes are evaluated in isolation, and thus are not amenable to fault interactions. Therefore, the simultaneous approach has precision ($\frac{TP}{TP+FP}$) and specificity ($\frac{TN}{TN+FP}$) intrinsically of 100%. Finally, the *fault recall* can be computed as $\frac{TP}{TP+FN}$, which should be as close as possible to 100%, i.e., FN are as few as possible.

4 Experimental evaluation

4.1 Experimental set-up

Evaluated DNNs and platform: We experiment with the Torchvision pre-trained quantized models [28] for CNNs, and two vision transformers (ViT) models provided in [17]. The models have different sizes and accuracies, as depicted in Table 1. Other relevant full precision benchmarks (e.g., [2]) can also be transparently integrated. Nonetheless, we are currently focused on quantized models as we evaluate injections both at the SW and RTL levels, applicable to state-of-the-art accelerators using an 8-bit integer format. Note that for RTL simulation, a single tile is simulated in Gemmini, which suffices for **transient fault injection**, so the time spent in RTL is small w.r.t. the full model forward pass, and the injection time in RTL is not much higher than in SW. Without loss of generality, a subset of 640 randomly selected inputs from the ImageNet validation set is used. Our platform is a 96-core AMD EPYC 7443 server processor @2.8GHz for the CNN models. The ViT models are experimented with on an NVIDIA A40 GPU.

Fault Injection: Fault injection for CNNs targets convolutional layers by instrumenting the layer's forward pass. For the ViT models, we target all matmul-related tasks, encompassing: 1) the computation of the Query(Q), Key(K) and Value(V) matrices; 2) the attention blocks; 4) a multi-layer perceptron module operating at the output of the attention; and 4) the classification head.

We performed Single Event Upsets (SEU) at software and RTL levels, injecting 500 faults per layer and per input, totaling 42M faults to ensure statistical significance [24]. The experiments employ a single bit-flip fault model, allowing us to analyze the resulting interactions. For CNN software injections, we perform a bit-flip in the layer activation by uniformly selecting a random layer, filter,

Table 1. Evaluated quantized models and fault injection time of the sequential approach at RTL (Gemmini∗) and SW levels.

Quantized model	Accuracy (Top-1)	Params	Inj. Time (RTL)	Inj. Time (SW)
MobileNetV2	71.6%	3.50M	2.80h	2.44h
DeiT-T	72.24%	5.00M	4.93h	4.76h
GoogLeNet	69.8%	6.60M	3.10h	2.73h
ShuffleNetX20	75.3%	7.40M	12.6h	12.1h
ResNet18	69.4%	11.7M	1.80h	1.66h
DeiT-S	80.1%	22.0M	8.21h	7.93h
ResNet50	80.2%	25.6M	7.00h	6.64h
InceptionV3	77.1%	27.2M	7.90h	8.00h
ResNeXt64	82.8%	83.5M	24.6h	23.0h
ResNeXt32	82.5%	88.8M	28.2h	25.3h

∗A single tile is simulated in Gemmini during **transient fault injection**, so the time spent in RTL is small w.r.t the full model forward pass.

channel, and bit position. For the ViT, SW injection targets the activations of random layers, attention blocks, attention heads inside the blocks, positions in the classification head and a target bit.

RTL injections are performed using the Gemmini systolic array as a case study [11]. To enable RTL injection, we built an injection framework based on PyTorch, integrated with Gemmini. At runtime, PyTorch performs calls to Gemmini in the layers forward pass. For the CNNs, the convolution layers are implemented as matrix multiplications after applying the `im2col` procedure to the input tensors. Two tiles, one for the activations and the other for the layer weights, are offloaded to Gemmini to perform matrix multiplication. For the ViT model, all matmul-related kernels and linear layers are deployed directly to Gemmini for injection. Transient single-bit faults are injected in the array registers by uniformly selecting a Processing Element (PE) and a PE bit position during multiplication. Once the RTL computation finishes, the PE outputs are flushed from the array and mapped back to the PyTorch output layers.

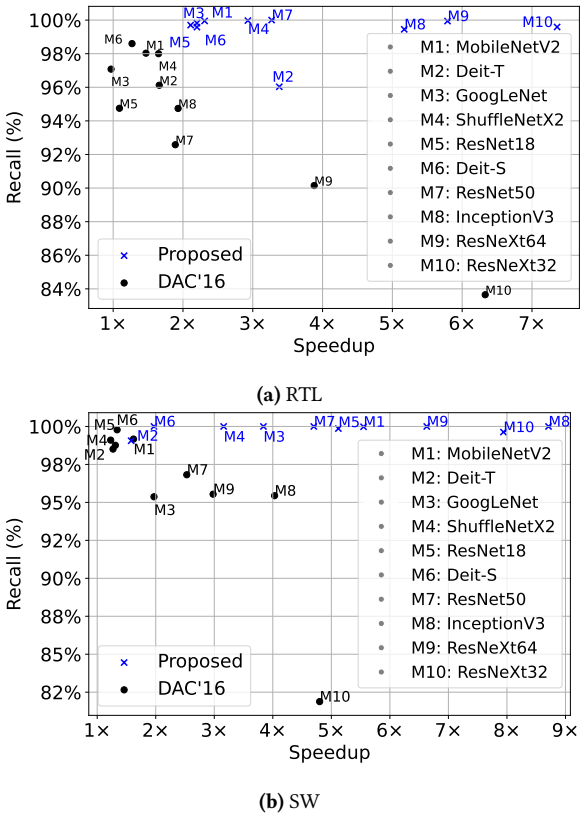
Comparisons: We evaluate our approach in fault recall, fault criticality, speedup, and scalability by comparing: (i) the proposed simultaneous injection methodology applicable for DNNs (*Proposed*); (ii) the conventional fault injection approach (*Sequential*); and (iii) a simultaneous approach oblivious to fault-to-fault interactions, such as [10] (*DAC'16*). For better clarity, the x-axis of plots are sorted by model size, when applicable.

4.2 Evaluation results

Fault Recall and Fault Criticality: The y-axis of Fig. 3 shows the fault recall for the proposed methodology and *DAC'16* for the SW and RTL levels. Overall, we observe that our approach achieves high fault recall for all evaluated models, i.e., on avg. 99.78% for RTL and 99.94% for SW. On the contrary, existing simultaneous approaches oblivious to fault interactions, *DAC'16*, usually degrade fault recall, i.e., avg. 94.1% for RTL and 95.76% for SW, down to 83.7% in RTL and 81.9% in SW for ResNeXt32.

Fig. 4 compares the ResNet50 fault criticality distribution for RTL (measured as AVF), and for SW (measured as PVF), for the three approaches. To have a high fault recall, the distribution should be as close as possible to the sequential approach. The results show that the distribution of the proposed approach overlaps with that of the sequential approach. Contrarily, *DAC'16* has bins where a significant number of highly critical faults is missing, e.g., up to 83% (88%) of the faults with criticality around 1.5% are not detected in RTL (SW). To complement this analysis, we use the Wasserstein distance as a similarity metric of the distributions of the proposed approach and *DAC'16* compared to the sequential one, shown in Table 2. Small distances indicate that the criticality distribution is closer to the sequential distribution. For *DAC'16*, the average distance across all models is around 0.154 (indicating a very different distribution), while our approach significantly reduces it to 0.034, leading to an around 4.5× improvement.

Speedup analysis: The x-axis of Fig. 3 shows the injection campaign speedup of the proposed approach and *DAC'16* compared to sequential. Note that simultaneous fault injection benefits both approaches. However, as our approach prunes inputs, while the exploration moves down the tree, it achieves significant speedup compared to *DAC'16*. Pruning non-critical and confident inputs reduces the average time per node visit, providing an overall gain, even when exploring the complete tree. On average, our approach reduces the time spent per node visit by a factor of 5.12× compared

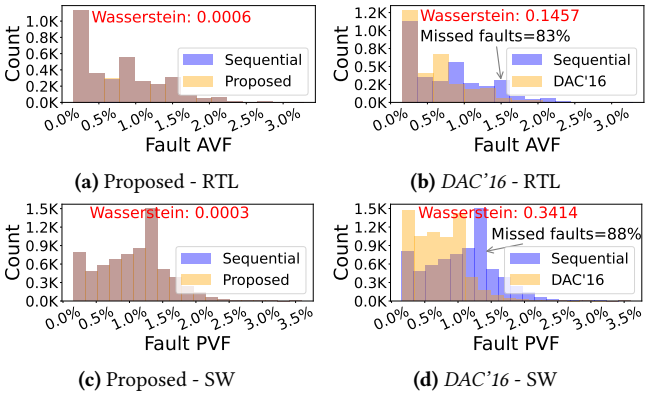


(a) RTL

(b) SW

Figure 3. Fault recall and speedup for RTL and SW level.

to *DAC'16*. Our strategy might increase the number of node visits in order to capture the critical faults that escape using the *DAC'16* method, thereby improving the accuracy of the reliability assessment. On average, we achieve an injection speedup of 3.82× for RTL and 5.29× for SW over the sequential case. *DAC'16* achieves an average speedup of 2.3× for RTL and 3.8× for SW. Hence, we obtain a 66.4% (2.18×) reduction in RTL (SW) injection time, while improving fault recall and AVF, compared to *DAC'16*.



(a) Proposed - RTL

(b) *DAC'16* - RTL

(c) Proposed - SW

(d) *DAC'16* - SW

Figure 4. Fault criticality distribution (ResNet50) for the proposed and DAC'16 approaches (yellow) compared to the sequential approach (blue). The brown color means the two distributions overlap.

Table 2. Wasserstein similarity metric (Lower is better).

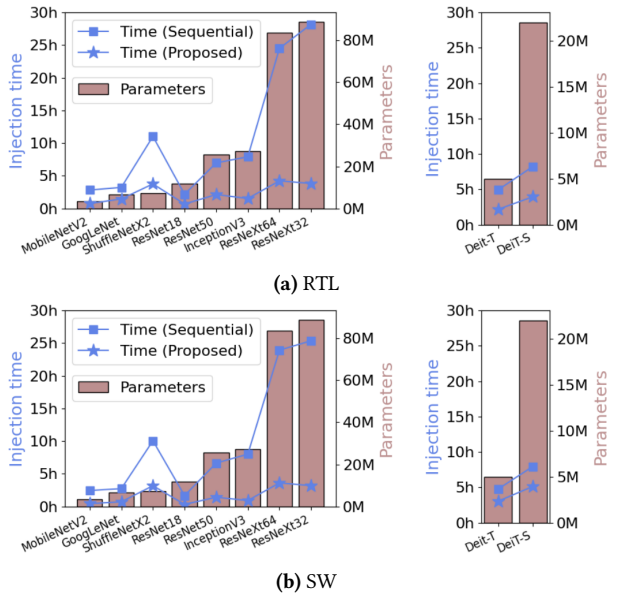
Model	RTL		SW	
	DAC'16	Proposed	DAC'16	Proposed
MobileNetV2	5.90×10^{-2}	5.30×10^{-2}	9.32×10^{-2}	8.76×10^{-2}
DeiT-T	1.71×10^{-3}	1.80×10^{-3}	$2.06x\times10^{-2}$	1.56×10^{-2}
GoogLeNet	5.60×10^{-2}	6.76×10^{-3}	2.32×10^{-1}	4.83×10^{-3}
ShuffleNetX2	6.13×10^{-2}	3.25×10^{-3}	4.72×10^{-1}	1.39×10^{-1}
ResNet18	5.81×10^{-2}	3.24×10^{-2}	2.38×10^{-1}	1.22×10^{-1}
DeiT-S	1.18×10^{-3}	1.19×10^{-3}	3.40×10^{-2}	1.27×10^{-2}
ResNet50	1.46×10^{-1}	5.91×10^{-4}	3.41×10^{-1}	2.85×10^{-4}
InceptionV3	8.33×10^{-2}	1.99×10^{-2}	2.59×10^{-1}	2.75×10^{-2}
ResNeXt64	1.48×10^{-1}	8.06×10^{-3}	3.73×10^{-1}	1.33×10^{-2}
ResNeXt32	1.64×10^{-1}	1.24×10^{-2}	1.52×10^{-1}	3.60×10^{-2}

Scalability Analysis: Fig. 5 shows the time in hours (left axis) at RTL and SW level of the proposed and sequential approaches against the model size, in millions of parameters (right axis). Overall, we observe similar trends for RTL and SW level injections.

As expected, the time required to perform reliability assessment using the sequential approach in general grows significantly as the model size increases. Some exceptions exist, as the fault injection time depends on the model inference time, and some models are faster even with more parameters. The proposed approach leverages the model's intrinsic ability to mask faults. The higher the fraction of masked faults, the higher the speedup, as more child nodes are pruned from the tree. As shown in Fig. 6, larger models tend to be less sensitive to faults (for CNNs). The left y-axis shows the percentage of injections during inference that result in misclassifications. For the same task/dataset, one reason larger models are more robust is that the number of redundant, non-vulnerable parameters increases with model size. This is in line with previous research showing that a few neurons are essential for correct classification (mostly due to model over-parameterization) [1]. As a consequence, this increases the pruning ratio of nodes, as they have a lower probability of being critical, leading to sublinear growth in injection time for larger models. The ViT models, however, are not directly comparable to the CNNs due to very different network structures. The encoding layer of the Q, K and V matrices, for example, is very sensitive to faults, leading to an increased vulnerability even when compared to larger CNN models.

5 Conclusions

We address the applicability and scalability issues of simultaneous fault injections in DNNs, showing that simultaneous faults interact and cause false negatives that undermine reliability assessment accuracy. Building on [10], we propose a methodology that explicitly accounts for these DNN-specific fault interactions, mitigating them to achieve near-optimal fault recall while preserving accuracy across SW and RTL abstraction levels. We introduce a performance-tuning mechanism that prunes non-critical, high-confidence inputs, reducing experimental time without sacrificing representativeness. Extensive experiments on quantized DNNs demonstrate efficient scaling with model size, reaching 100% fault recall (min 99.5%) at both RTL and SW levels, and reducing RTL (SW) injection time by 3.82× (5.29×) on average compared to conventional approaches, with sub-linear time growth as DNN size increases.



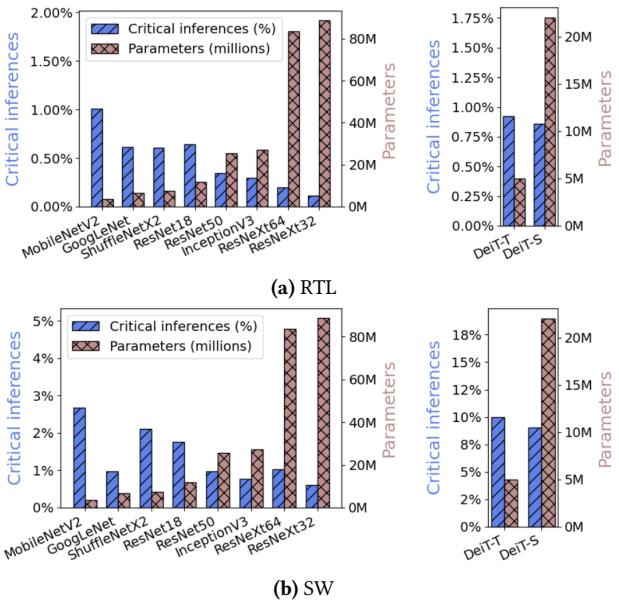
(a) RTL

(b) SW

Figure 5. Injection time trends for proposed and sequential approaches. The injection time for the proposed approach grows sub-linearly in relation to the model size.

Acknowledgment

This work has been funded by the French National Research Agency through the RE-TRUSTING project ANR-21-CE24-0015 and by the European Union via the European Defence Fund project ARCHYTAS under grant number 101167870. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Comission. Neither the European Union nor the granting authority can be held responsible for them.



(a) RTL

(b) SW

Figure 6. Percentage of critical runs and the size of model in terms of parameters.

References

[1] David Bau et al. 2020. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences* 117, 48 (Sept. 2020), 30071–30078. doi:10.1073/pnas.1907375117

[2] Cristiana Bolchini et al. 2025. Benchmark Suite for Resilience Assessment of Deep Learning Models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2025), 1–1. doi:10.1109/TCAD.2025.3578297

[3] Cristiana Bolchini, Luca Cassano, Antonio Miele, and Alessandro Toschi. 2023. Fast and Accurate Error Simulation for CNNs Against Soft Errors. *IEEE Trans. Comput.* 72, 4 (April 2023), 984–997. doi:10.1109/TC.2022.3184274

[4] Nandhini Chandramoorthy et al. 2019. Resilient Low Voltage Accelerators for High Energy Efficiency. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 147–158. doi:10.1109/HPCA.2019.00034

[5] Zitao Chen, Niranjhana Narayanan, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2020. TensorFI: A Flexible Fault Injection Framework for TensorFlow Applications. arXiv:2004.01743 [cs.DC] https://arxiv.org/abs/2004.01743

[6] Leonardo Alexandrino de Melo et al. 2025. MicroFI: TensorFlow Lite based Fault Injection Framework for Microcontrollers. In *2025 IEEE 43rd VLSI Test Symposium (VTS)*. 1–7. doi:10.1109/VTS65138.2025.11022806

[7] Harish Dattatraya Dixit et al. 2022. Detecting silent data corruptions in the wild. arXiv:2203.08989 [cs.AR] https://arxiv.org/abs/2203.08989

[8] Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, and Sriram Sankar. 2021. Silent Data Corruptions at Scale. arXiv:2102.11245 [cs.AR] https://arxiv.org/abs/2102.11245

[9] Rhea Dutta et al. 2025. Hardware Sentinel: Protecting Software Applications from Hardware Silent Data Corruptions. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 482–497. doi:10.1145/3676641.3716258

[10] Mojtaba Ebrahimi et al. 2016. Fault injection acceleration by simultaneous injection of non-interacting faults. In *Proceedings of the 53rd Annual Design Automation Conference* (Austin, Texas) *(DAC '16)*. Association for Computing Machinery, New York, NY, USA, Article 25, 6 pages. doi:10.1145/2897937.2898023

[11] Hasan Genc et al. 2022. *Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration.* IEEE Press, 769–774. https://doi.org/10.1109/DAC18074.2021.9586216

[12] Siva Kumar Sastry Hari et al. 2017. SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 249–258. doi:10.1109/ISPASS.2017.7975296

[13] Yi He et al. 2023. Understanding and Mitigating Hardware Failures in Deep Learning Training Systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 70, 16 pages. doi:10.1145/3579371.3589105

[14] Yi He, Prasanna Balaprakash, and Yanjing Li. 2020. FIdelity: Efficient Resilience Analysis Framework for Deep Learning Accelerators. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 270–281. doi:10.1109/MICRO50266.2020.00033

[15] Peter H. Hochschild, Paul Turner, Jeffrey C. Mogul, Rama Govindaraju, Parthasarathy Ranganathan, David E. Culler, and Amin Vahdat. 2021. Cores that don't count. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Ann Arbor, Michigan) *(HotOS '21)*. Association for Computing Machinery, New York, NY, USA, 9–16. doi:10.1145/3458336.3465297

[16] Xun Jiao et al. 2017. An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 945–950. doi:10.1109/ICCAD.2017.8203882

[17] Zhikai Li and Qingyi Gu. 2023. I-ViT: Integer-only Quantization for Efficient Vision Transformer Inference. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 17019–17029. doi:10.1109/ICCV51070.2023.01565

[18] Abdulrahman Mahmoud, Siva Kumar Sastry Hari, Christopher W. Fletcher, Sarita V. Adve, Charbel Sakr, Naresh Shanbhag, Pavlo Molchanov, Michael B. Sullivan, Timothy Tsai, and Stephen W. Keckler. 2021. Optimizing Selective Protection for CNN Resilience. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. 127–138. doi:10.1109/ISSRE52982.2021.00025

[19] S.S. Mukherjee et al. 2003. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* 29–40. doi:10.1109/MICRO.2003.1253181

[20] George Papadimitriou and Dimitris Gizopoulos. 2021. Demystifying the system vulnerability stack: transient fault effects across the layers. In *Proceedings of the 48th Annual International Symposium on Computer Architecture* (Virtual Event, Spain) *(ISCA '21)*. IEEE Press, 902–915. doi:10.1109/ISCA52012.2021.00075

[21] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. 2018. Ares: a framework for quantifying the resilience of deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference* (San Francisco, California) *(DAC '18)*. Association for Computing Machinery, New York, NY, USA, Article 17, 6 pages. doi:10.1145/3195970.3195997

[22] Paolo Rech. 2024. Artificial Neural Networks for Space and Safety-Critical Applications: Reliability Issues and Potential Solutions. *IEEE Transactions on Nuclear Science* 71, 4 (2024), 377–404. doi:10.1109/TNS.2024.3349956

[23] Rubens Luiz Rech and Paolo Rech. 2022. Reliability of Google's Tensor Processing Units for Embedded Applications. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 376–381. doi:10.23919/DATE54114.2022.9774600

[24] A. Ruospo et al. 2023. Assessing Convolutional Neural Networks Reliability through Statistical Fault Injections. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1–6. doi:10.23919/DATE56975.2023.10136998

[25] Annachiara Ruospo, Ernesto Sanchez, Lucas Matana Luza, Luigi Dilillo, Marcello Traiola, and Alberto Bosio. 2023. A Survey on Deep Learning Resilience Assessment Methodologies. *Computer* 56, 2 (Feb. 2023), 57–66. doi:10.1109/MC.2022.3217841

[26] Fernando Fernandes dos Santos et al. 2019. Analyzing and Increasing the Reliability of Convolutional Neural Networks on GPUs. *IEEE Transactions on Reliability* 68, 2 (2019), 663–677. doi:10.1109/TR.2018.2878387

[27] Vilas Sridharan and David R. Kaeli. 2008. Quantifying software vulnerability. In *Proceedings of the 2008 Workshop on Radiation Effects and Fault Tolerance in Nanometer Technologies* (Ischia, Italy) *(WREFT '08)*. Association for Computing Machinery, New York, NY, USA, 323–328. doi:10.1145/1366224.1366225

[28] TorchVision Contributors. 2025. TorchVision: PyTorch's Computer Vision Library. https://docs.pytorch.org/vision/main/models#quantized-models. Accessed: 2025-11-17.

[29] Abhishek Tyagi et al. 2024. Thales: Formulating and Estimating Architectural Vulnerability Factors for DNN Accelerators. arXiv:2212.02649 [cs.AR] https://arxiv.org/abs/2212.02649

[30] Alessandro Veronesi et al. 2024. Cross-Layer Reliability Analysis of NVDLA Accelerators: Exploring the Configuration Space. In *Proceedings of the 29th IEEE European Test Symposium 2024*.

[31] Shaobu Wang, Guangyan Zhang, Junyu Wei, Yang Wang, Jiesheng Wu, and Qingchao Luo. 2023. Understanding Silent Data Corruptions in a Large Production CPU Population. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 216–230. doi:10.1145/3600006.3613149

[32] Xiang-Xiu Wu et al. 2020. Accuracy Tolerant Neural Networks Under Aggressive Power Optimization. In *2020 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 774–779. doi:10.23919/DATE48585.2020.9116475

[33] Ussama Zahid et al. 2020. FAT: Training Neural Networks for Reliable Inference Under Hardware Faults. arXiv:2011.05873 [cs.LG] https://arxiv.org/abs/2011.05873

[34] Jeff Zhang et al. 2018. Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators. In *Proceedings of the 55th Annual Design Automation Conference* (San Francisco, California) *(DAC '18)*. Association for Computing Machinery, New York, NY, USA, Article 19, 6 pages. doi:10.1145/3195970.3196129

[35] Jeff Jun Zhang et al. 2018. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In *2018 IEEE 36th VLSI Test Symposium (VTS)*. 1–6. doi:10.1109/VTS.2018.8368656