POLITECNICO DI TORINO
Repository ISTITUZIONALE

Implicit Activation Sparsity for Fault-Tolerant CNNs: A Training-Dynamics Perspective

Original
Implicit Activation Sparsity for Fault-Tolerant CNNs: A Training-Dynamics Perspective / Bellarmino, N., Bosio, A., Ruospo, A., Ernesto, S., Cantoro, R.. - IEEE International Conference on Omni-Layer Intelligent Systems (COINS):(In corso di stampa). (IEEE International Conference on Omni-Layer Intelligent Systems (COINS) Bologna, Italy 7-9 September 2026).

Availability:
This version is available at: 11583/3016154 since: 2026-09-30T11:32:25Z

Publisher:
IEEE

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©9999 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

16 November 2026

Implicit Activation Sparsity for Fault-Tolerant CNNs: A Training-Dynamics Perspective

Nicolò Bellarmino*, Alberto Bosio†, Annachiara Ruospo*, Ernesto Sanchez*, Riccardo Cantoro*

*CAD Research Group,
DAUIN, Politecnico di Torino, Italy
{name.surname}@polito.it

† Univ Lyon, ECL, INSA Lyon, CNRS, UCBL,
CPE Lyon, INL, UMR5270, France
alberto.bosio@ec-lyon.fr

***Abstract*—Convolutional Neural Networks (CNNs) are increasingly deployed in safety-critical embedded systems, where reliability against hardware faults is crucial. Traditional fault-tolerance techniques rely on redundancy or fault-aware training, both of which introduce computational or design overhead. This paper presents a zero-cost approach to improve CNN fault tolerance by exploiting implicit sparsity induced through training dynamics. We show that tuning the learning rate during training naturally promotes sparse internal representations that act as *fault masks*, whose masking probability can be modeled statistically, enhancing resilience without explicit fault injection. We derive and experimentally validate a quantitative model linking sparsity and fault masking, demonstrating a strong linear correlation ($r = 0.91$) across four CNN architectures and two standard datasets. Highly sparse models achieve up to 90% fault masking relative to standard training, while dynamic batch-size co-scheduling mitigates accuracy loss and preserves robustness. Beyond algorithmic effects, sparsity *may reduce* Multiply–Accumulate (MAC) operations and switching activity in hardware accelerators, potentially yielding energy efficiency benefits. This work reframes CNN reliability as a *property of optimization dynamics* rather than a post-training add-on.**

***Index Terms*—Activation Sparsity, Fault Tolerance, Reliability, Convolutional Neural Networks (CNNs), Training Dynamics, Fault Injection, Hardware Accelerators, Deep Learning, Optimization Dynamics, Fault Resilience.**

I. Introduction

Deep learning models, particularly Convolutional Neural Networks (CNNs), are at the core of modern computer vision systems deployed in safety-critical domains such as autonomous driving, medical diagnostics, and aerospace control [1]–[3]. In such contexts, even transient hardware faults during inference can cause catastrophic system-level failures [4], [5]. Ensuring that CNNs remain reliable under hardware faults has therefore become a central research challenge. Two main strategies dominate this landscape. The first relies on *redundancy-based hardening* at the architectural or system level [6], [7], which increases area, power, and latency. The second approach, known as *Fault-Aware Training (FAT)* [8]–[10], injects faults during training so that the network learns to tolerate them. While effective, FAT substantially increases training time, complexity, and computational cost, making it impractical for frequent model updates or on-device retraining.

In this work, we explore a third, orthogonal direction: improving reliability by exploiting the **training dynamics** themselves. We show that simple adjustments to the learning rate naturally induce *implicit activation sparsity* in CNN representations, which in turn enhances fault masking. This effect arises without architectural changes or explicit regularization. Our central hypothesis is that activation sparsity reduces error propagation from weight or activation faults, as inactive connections act as natural fault isolators. We formalize this intuition through a statistical model that describes the probability of fault masking as a function of activation sparsity. Under small perturbations, we show that the expected masking probability grows approximately linearly with sparsity, providing a simple yet effective quantitative predictor of reliability. We refer to this approach as *implicit sparsity training*. Unlike redundancy or FAT-based methods, it requires no additional computations, fault models, or retraining steps, making it inherently scalable. Moreover, sparsity has potential hardware benefits: zero-valued activations *may reduce* the number of multiply–accumulate (MAC) operations, memory accesses, and switching activity in digital accelerators [11], [12], suggesting possible energy efficiency improvements on sparse-aware platforms. Our contributions are threefold:

- We establish a quantitative link between training-induced sparsity and CNN fault resilience across multiple architectures and datasets, supported by a statistical model that predicts masking probability as a linear function of sparsity.
- We provide an analytical interpretation of sparsity-driven fault masking and propose normalized resilience metrics to remove baseline accuracy bias.
- We discuss the broader implications of sparsity-induced robustness, connecting software-level learning dynamics to system-level reliability and efficiency.

This work positions reliability as an *emergent property of learning dynamics*, reframing CNN robustness as a controllable outcome of the optimization process rather than an external design constraint. The remainder of this paper is organized as follows. Section II reviews related work on CNN reliability

979-8-3195-0489-0/26/$31.00 ©2026 IEEE

analysis. Section III introduces the proposed method, while Sections IV–V describe the experimental setup and results. Finally, Section VI concludes the paper and outlines future research directions.

II. Background and Related Work

A. CNN Reliability and Fault Injection Models

Hardware faults in AI accelerators can be broadly categorized as permanent or transient. Permanent faults arise from physical degradation or manufacturing defects and are typically modeled using the *stuck-at fault* abstraction, where a bit or signal is permanently fixed at logic 1 or 0 [13]. Transient faults (soft errors) are temporary disruptions caused by radiation or electromagnetic interference, often modeled as random bit flips in memory or registers [14]–[16].

Recent systematic reviews have classified hardware reliability assessment methods for DNNs into fault-injection (FI), analytical, and hybrid approaches, highlighting that FI remains the dominant methodology but also that platform-dependent effects may significantly affect reliability estimates [14], [17]–[19].

FI artificially emulates hardware faults at different abstraction levels. Simulation-based FI operates on high-level or architectural models. Platform-based FI uses hardware emulation or FPGA prototypes. Radiation-based FI exposes real hardware to fault-inducing environments. The resulting inference outcomes are categorized as *masked*, *non-critical*, or *critical (silent data corruption)* depending on their impact on the final output [15]. While FI-based evaluation is essential for understanding CNN fault behavior, it does not inherently improve resilience—it only measures it. Most existing FI frameworks are resource-intensive, require specialized hardware, and offer limited insight into the underlying learning mechanisms that influence resilience. More recently, model-level fault tolerance has been questioned as a sufficient proxy for accelerator-level reliability, since software-level robustness may not always correlate with hardware-level behavior and may introduce non-trivial efficiency side effects [18].

B. CNN Hardening Approaches

Existing fault-tolerance strategies for CNNs fall into two broad categories: *post-training hardening* and *fault-aware training (FAT)*. Post-training methods introduce structural or temporal redundancy to detect and mask faults. Examples include selective hardening of vulnerable layers [7], [20], duplication-based fault masking, or architectural redesigns to enhance error resilience [6], [21]. While effective, these approaches increase area, power, and latency, making them unsuitable for edge deployment.

FAT integrates fault effects into the learning process [8]–[10]. By injecting faults during training, the network learns to compensate for their presence. FAT improves robustness but incurs high computational cost and retraining time. Furthermore, FAT requires predefined fault models and hardware fault profiles, limiting its adaptability to real-world conditions.

Despite these advances, both categories rely on either explicit redundancy or direct fault modeling. Neither exploits the inherent dynamics of the learning process as a reliability driver.

C. Training Dynamics and Implicit Regularization

The training dynamics of neural networks refer to the evolving behavior of weights, gradients, and activations during optimization [22]. Factors such as the learning rate, optimizer choice, and batch size govern how the network explores its parameter space and generalizes [23]. Recent studies have shown that training dynamics can implicitly regularize the network. For instance, high learning rates can induce *sparse feature representations* that improve generalization [24], [25], while specific optimizer–activation combinations such as Adam with ReLU naturally promote group sparsity [26]. Sparse representations deactivate portions of the network—neurons or channels that remain at or near zero during inference—which has been widely exploited for model compression and pruning [27]. Recent studies [28] have linked training dynamics to the *test domain*, showing that gradient regularization can improve fault observability in CNNs. This suggested that reliability may not only depend on architecture or redundancy, but also on how a network learns.

D. Gap and Motivation

Existing methods improve CNN reliability either by modifying the architecture (redundancy) or by injecting artificial faults (FAT). However, neither approach is *intrinsically adaptive* or *hardware-efficient*. The key unexplored question is whether reliability can emerge naturally from the learning process itself. Specifically, **can training dynamics—without external noise or redundancy—be tuned to produce representations that inherently tolerate faults?**

This work addresses that question by demonstrating that learning rate modulation induces sparsity, which in turn reduces the propagation of hardware faults through the network. Unlike FAT, the proposed approach achieves robustness without fault injection or architectural modifications. It maintains the standard training workflow, scales to any CNN architecture, and introduces no additional training overhead. By reframing fault resilience as an emergent property of *learning dynamics*, our method bridges the gap between optimization behavior and *hardware reliability*.

III. Proposed Methodology

A. Motivation and Core Hypothesis

Prior studies have shown that higher learning rates during training can implicitly induce sparse weights and activation distributions due to stronger regularization effects and more stochastic gradient dynamics [25], [26]. Such sparse representations deactivate subsets of neurons or channels, effectively pruning redundant signal paths. We hypothesize that this emergent sparsity provides a natural *fault isolation mechanism*. When a fault corrupts a weight or activation belonging to an inactive (zero-valued) pathway, its impact on the output is

negligible. Hence, sparsity acts as a **statistical fault mask**, reducing the propagation energy of fault-induced perturbations.

Formally, consider a linearized layer $y = Wx$ affected by small fault-induced perturbations. The first-order output deviation can be written as

$$\Delta y \approx W\,\Delta x + \Delta W\,x, \tag{1}$$

where W and x are the nominal weights and inputs, while ΔW and Δx denote perturbations affecting weights and activations, respectively. In the fault-injection campaign considered in this work, faults are injected in the stored weights. Therefore, the dominant perturbation term is $\Delta W x$, while $\Delta x = 0$ for the injected fault model. If a fraction s of activation channels is inactive, only the remaining active fraction $(1 - s)$ can propagate weight perturbations to the layer output. Under the standard assumptions that fault locations are independent of the input activations, that perturbations are zero-mean, and that the second moment of non-zero activations remains comparable across configurations, the expected squared output deviation scales as

$$\mathbb{E}_{x,\Delta W}\left[\|\Delta y\|_2^2\right] = \mathbb{E}_{x,\Delta W}\left[\|\Delta W x\|_2^2\right] \propto (1-s)\,\mathbb{E}_{\Delta W}\left[\|\Delta W\|_F^2\right]. \tag{2}$$

Here, $\mathbb{E}[\cdot]$ denotes the statistical expectation over input samples, fault locations, and fault values. We refer to $\|\Delta y\|_2^2$ as perturbation energy, meaning the squared magnitude of the output deviation. Architecture-dependent constants, layer dimensionality, and the average squared magnitude of active activations are absorbed into the proportionality term. Thus, Eq. (2) does not introduce a separate energy operator; it simply states that activation sparsity reduces the expected magnitude of fault-induced output perturbations by removing inactive propagation paths. A fault is considered *masked* when its effect on the output remains below a decision threshold τ, so that the predicted class does not change. Assuming the perturbations follow a zero-mean distribution with variance proportional to the energy in (2), the masking probability can be expressed as

$$M = P(|\Delta y| < \tau) = F\left(\frac{\tau}{\sigma_{\Delta y}}\right), \tag{3}$$

where $F(\cdot)$ is the cumulative distribution function of the perturbation amplitude and $\sigma_{\Delta y}^2 \propto (1 - s)$. Here, $\sigma_{\Delta y}$ denotes the standard deviation of the perturbation, and the variance $\sigma_{\Delta y}^2$ scales proportionally to $(1 - s)$. The function $F(\cdot)$ is used as a generic cumulative distribution function and does not require assuming a specific normal distribution.

Expanding $F(\cdot)$ around its nominal operating point yields a first-order approximation:

$$M(s) \approx M_0 + \alpha s, \tag{4}$$

where M_0 represents the baseline masking ratio in dense networks and α quantifies how strongly robustness increases with sparsity.

The parameters M_0 and α are therefore empirical coefficients: M_0 captures the baseline masking capability of a dense or weakly sparse network, while α captures the sensitivity of masking probability to activation sparsity.

Intuitively, Equation (2) describes how sparsity reduces the expected energy of fault perturbations, while the linear model $M(s)$ translates this attenuation into a statistical increase in masking probability. This connection links the physical attenuation of fault effects to the empirical masking behavior observed in Section V-A.

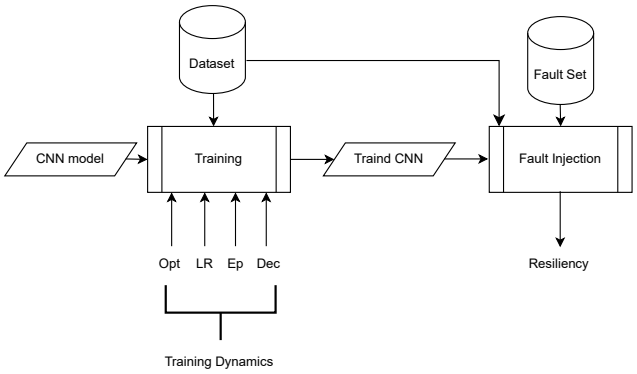


Fig. 1: Overview of the proposed implicit sparsity framework.

B. Framework Overview

Fig. 1 summarizes the proposed workflow. The procedure consists of four steps:

1) **Training:** Each CNN is trained multiple times while varying learning rate η, optimizer, and batch-size schedule.
2) **Sparsity Profiling:** After training, sparsity s is computed from the penultimate-layer activations as the fraction of near-zero channels.
3) **Fault Injection:** Trained models undergo a SFI campaign where random bit-level stuck-at faults emulate permanent hardware errors [14].
4) **Resilience Metrics:** Faults are categorized as masked, non-critical, or critical [15], and accuracy degradation (*regret*) is recorded.

C. Learning Dynamics as a Reliability Lever

The central control variable in this study is the **learning rate (η)**. Increasing η amplifies gradient noise and parameter oscillations, which empirically leads to sparse feature activation [24]. However, excessive learning rates may cause instability or loss of accuracy. We therefore explore a trade-off space:

$$\text{Robustness Gain } G_r(\eta)\ \uparrow\ \text{ vs. Accuracy Loss } L_a(\eta)\ \uparrow.$$

Empirically, moderate η values (e.g., 10^{-2}–10^{-1}) yield optimal fault-masking ratios while keeping the accuracy loss below 2%. To mitigate over-regularization, we adopt a *dynamic batch-size schedule* that complements learning-rate decay and preserves both convergence stability and induced sparsity. Details and motivations are given in Section IV-E. Pseudocode in Algorithm 1 outlines the full training process.

Algorithm 1 Implicit Sparsity Training for CNN Reliability

Input: Dataset $\mathcal{D}$, CNN model $\mathcal{M}$, base learning rate η_0, decay rate γ, batch schedule $\{B_t\}$
for epoch $t = 1$ to T **do**
    Update learning rate: $\eta_t = \eta_0 \cdot e^{-\gamma t}$
    Update batch size: $B_t = B_{t-1} \cdot \gamma_B$
    Train $\mathcal{M}$ on $\mathcal{D}$ using Adam or SGD (η_t, B_t)
end for
Measure activation sparsity s in the penultimate layer.
Perform fault injection on weights to evaluate masking ratio $M(s)$.
Output: Trained model $\mathcal{M}^*$, sparsity level s, fault-masking metrics.

D. Advantages over Fault-Aware Training

Compared to FAT and redundancy-based hardening techniques, implicit sparsity training offers the following benefits:

- **No explicit fault model:** Robustness emerges naturally, without assuming specific fault types or locations.
- **Zero training overhead:** No fault injection or re-training phase is required.
- **Architectural transparency:** The network structure and training flow remain unchanged, making the approach easily applicable to existing CNNs.

This formulation elevates fault tolerance from a post-design feature to an inherent outcome of the training-loop optimization process. The proposed method differs from FAT in the mechanism by which robustness is induced. FAT explicitly injects faults during training, so the optimization process directly adapts the weights to a predefined fault model, fault rate, and injection policy. In contrast, our method never exposes the network to faulty computations during training: robustness emerges indirectly from the sparse activation patterns induced by the learning-rate and batch-size dynamics. Therefore, the method is less fault-model-specific than FAT, but also less targeted. A controlled same-architecture comparison with FAT is outside the scope of this work and is left as future work; the comparison reported here should be interpreted as methodological rather than as a direct performance ranking.

IV. Experimental Setup and Evaluation

A. Experimental Configuration

We evaluate four CNN architectures representative of modern deployment scales:

- **ResNet-9** and **ResNet-18** [29]: deep residual models with different depth-to-parameter ratios;
- **MobileNetV2** [30], [31]: an efficient lightweight architecture for embedded inference;
- **HCGNet-A2** [32]: a gated hybrid-connectivity CNN optimized for dense classification.

ResNet-9, ResNet-18, and MobileNetV2 are trained on CIFAR-10, whereas HCGNet-A2 is trained on CIFAR-100.

For each model, we train variants by varying the learning rate $\eta \in \{10^{-3}, 10^{-2}, 10^{-1}\}$, while keeping other parameters constant unless explicitly noted. The optimizer is Adam for ResNet and MobileNet, and SGD for HCGNet. Batch sizes are set to 64 by default with optional batch-size scheduling (Section IV-E). Training is conducted for 100 epochs using CIFAR-10 and CIFAR-100 [33] with data augmentation (random crop and rotation). The learning rate decays exponentially with a factor $\gamma = 0.09$ every five epochs. All networks were trained and evaluated in FP32 precision.

B. Sparsity Measurement

To measure the induced sparsity level s, we analyze activations in the penultimate layer (pre-classification embeddings). For each input image, the number of non-zero features is computed; a feature channel is considered inactive if its 99th-percentile activation magnitude across the training set is zero. The global sparsity level is thus defined as:

$$s = 1 - \frac{1}{N_f} \sum_{i=1}^{N_f} \mathbf{1}\{\text{activation}_i > 0\}, \tag{5}$$

where N_f is the number of embedding features. Figure 2 shows where we computed s. It is important to note that the sparsity analyzed in this work refers to the *activation sparsity* of feature embeddings rather than weight sparsity. Hence, the fault-masking behavior originates from the dynamic deactivation of neurons during training, not from pruning or zeroing of parameters.

The sparsity is measured at the penultimate layer because this layer provides an architecture-independent representation of the feature embedding directly used by the classifier. Earlier convolutional layers have architecture-specific spatial dimensions and channel structures, making cross-model comparison less direct. The penultimate embedding therefore offers a consistent proxy for representation-level sparsity. This choice does not imply that sparsity is absent or irrelevant in earlier layers; rather, it provides a common measurement point for correlating training dynamics with fault masking.

C. Fault Injection Campaign

An SFI [14] process was used to validate the robustness of the trained models. Permanent faults are modeled as bit-level stuck-at conditions applied to FP32 parameters in memory. While this abstraction operates above the hardware microarchitecture, it remains consistent with the memory-level fault manifestation discussed by Chatzopoulos *et al.* [34], who demonstrate that software-level fault injection can under- or overestimate Silent Data Corruption (SDC) rates by up to 4× if hardware timing and data-path dependencies are ignored. Our approach focuses on algorithmic fault masking rather than device-level propagation, making the bit-level abstraction suitable for analyzing resilience trends in CNNs. A total of about 16,640 random bit-flip faults are injected across all layers to estimate the resiliency metrics with an error margin $e = 0.01$ and confidence level of 0.99. Faults are distributed proportionally to each layer's parameter count, assuming the *single fault assumption* (no simultaneous faults). Each injected fault is categorized following the framework in [15]:

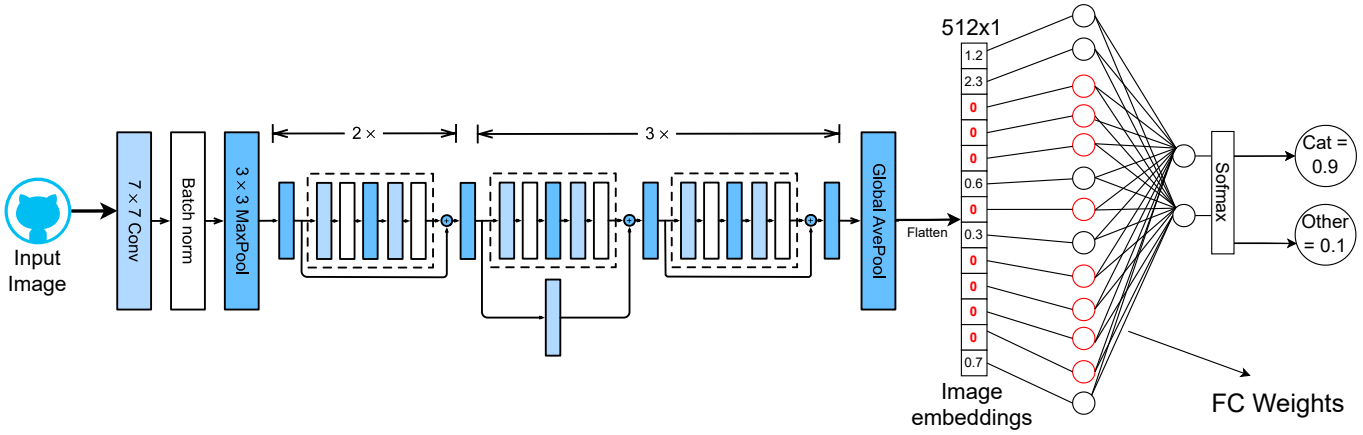


Fig. 2: Sparse ResNet18. Activation sparsity is computed in the penultimate layer, representing the feature embeddings extracted by the network, and not in the network weights. Inactive neurons (red circles) correspond to zero-valued activations that do not contribute to the output.

- **Masked:** Output identical to fault-free inference.
- **Non-critical:** Output differs in confidence but not in predicted class.
- **Critical:** Output misclassified (silent data corruption).

The *workload* [16], [17] for the SFI process is the whole *test set* for all the datasets under evaluation.

D. Evaluation Metrics

For each configuration, we report:

- **Clean Accuracy** (A_c): fault-free classification accuracy.
- **Faulty Accuracy** (A_f): accuracy under fault injection.
- **Normalized Resilience:**

$$R_n = \frac{A_f}{A_c}, \tag{6}$$

  which mitigates bias from reduced baseline accuracy
- **Sparsity Level** (s): proportion of inactive embedding channels.
- **Masked Fault Ratio** (M): proportion of injected faults that produce no output deviation.

The metrics are computed on the *test set* for each of the datasets under evaluation.

E. Batch-Size Scheduling and Dynamic Stability

To mitigate over-regularization and instability at high learning rates, we employ a **dynamic batch-size scheduling** strategy. The dynamics of SGD can be described by a stochastic differential equation whose *noise scale* is defined as $g = \eta N/B$, where η is the learning rate, N the dataset size, and B the batch size [35]. This noise term determines the stochasticity of the training dynamics: larger batches reduce gradient noise, while smaller batches increase it, influencing both convergence stability and the flatness of the resulting minima. In our framework, η and B are co-scheduled throughout training. We begin with a large learning rate and a large batch size to ensure stable convergence during the early stages, then progressively decrease both to increase gradient stochasticity as training progresses. Specifically, the batch size starts at 512 and decreases by a factor $\gamma_B = 0.8$ every ten epochs. This controlled noise amplification acts as an implicit regularizer, preventing over-smoothing and promoting neuron deactivation.

Our approach conceptually reverses the principle introduced by Smith *et al.* [36], who demonstrated that *increasing* the batch size during training can replicate the effects of learning-rate decay, leading to faster convergence and fewer parameter updates. While their goal was optimization efficiency, our objective is robustness enhancement. By progressively *decreasing* the batch size instead, we maintain a moderate level of gradient noise in later epochs, which counteracts excessive regularization and promotes activation sparsity.

Jointly decreasing both the learning rate and batch size thus establishes a balance between convergence stability and stochastic exploration. This *anti-annealing* regime yields flatter, sparser, and more fault-resilient minima.

V. Experimental Evaluation

A. Quantitative Results and Analysis

Table I summarizes the results obtained across all CNN architectures and training configurations. Increasing the learning rate consistently raises activation sparsity and fault masking, confirming that the learning dynamics directly affect robustness. For ResNet9, the proposed dynamic batch-size scheduling achieves the best trade-off between accuracy and fault tolerance, demonstrating that co-decaying the learning rate and batch size stabilizes training at high sparsity levels.

The observed trends are consistent across all networks:

- Fault masking M increases almost linearly with sparsity s, validating the analytical model introduced in Section III-A.
- Normalized resilience R_n remains above 97% for all configurations, demonstrating that improved robustness is not due to degraded baseline accuracy.
- The accuracy reduction associated with higher learning rates ($\leq$ 2%) is comparable to the drop reported in prior Fault-Aware Training (FAT) methods [8], [9], but

TABLE I: Effect of learning rate on performance and fault-type distributions across architectures.

LR	A_c (%)	A_f (%)	R_n (%)	M (%)	N_c (%)	C (%)	s (%)
MobileNetV2 (CIFAR-10)							
0.001	90.9	88.7	97.5	24.8	72.6	2.5	1.9
0.01	90.6	88.7	**97.9**	**58.5**	39.4	2.1	**85.1**
0.1	89.9	87.5	97.3	82.6	14.7	2.7	98.8
ResNet18 (CIFAR-10)							
0.001	92.7	90.5	97.6	37.9	59.8	2.3	0.0
0.01	91.6	90.0	98.3	67.2	31.0	1.8	22.5
0.1	90.5	89.5	**98.9**	**90.7**	8.2	1.1	**99.0**
ResNet9 (CIFAR-10)							
0.001	91.4	88.9	97.3	21.8	75.5	2.8	0.0
0.01	92.5	90.3	97.6	37.1	60.4	2.5	0.0
0.05 (BS Sc.)	90.7	88.8	**97.9**	**74.1**	23.8	2.1	**52.7**
HCGNet-A2 (CIFAR-100)							
0.01	82.4	81.4	98.8	23.0	75.7	1.3	0.0
0.05	79.6	78.6	98.8	35.8	63.0	1.2	15.2
0.1	75.3	74.9	**99.4**	**74.5**	24.9	0.6	**55.3**

Note: A_c/A_f denote clean/faulty accuracy; $R_n = 100 \cdot A_f/A_c$ is normalized resilience. M, N_c, and C denote masked, not-critical, and critical-fault ratios; s is activation sparsity. For ResNet9, the third row uses dynamic batch-size scheduling to illustrate the best accuracy–robustness balance.

TABLE II: ResNet9 ablation: impact of dynamic batch-size scheduling ("BS Sched.") compared to static batch sizes ("BS").

Configuration	A_c (%)	A_f (%)	R_n (%)	M (%)	s (%)
0.05 (BS 256)	88.6	86.9	98.1	79.7	62.5
0.05 (BS Sched.)	90.7	88.8	97.9	74.1	52.7
0.09 (BS 128)	85.9	83.9	97.6	86.0	92.2
0.095 (BS Sched.)	88.8	86.7	97.6	81.0	77.3
0.1 (BS Sched.)	88.5	86.4	97.7	82.4	83.6

achieved here without any explicit fault injection or redundancy mechanisms.

In our analysis, a fault is considered *masked* when it does not alter the output probability vector, not just the predicted class. As shown in Table I, increasing sparsity reduces the share of *Not-Critical* faults while increasing Masked ones, whereas *Critical* (SDC) faults remain stable or slightly decrease. This indicates that sparsification progressively transforms previously non-critical faults into fully masked events, consistent with the proposed model in which reduced perturbation energy ($E[\|\Delta y\|^2] \propto 1-s$) lowers the probability that small deviations affect the output.

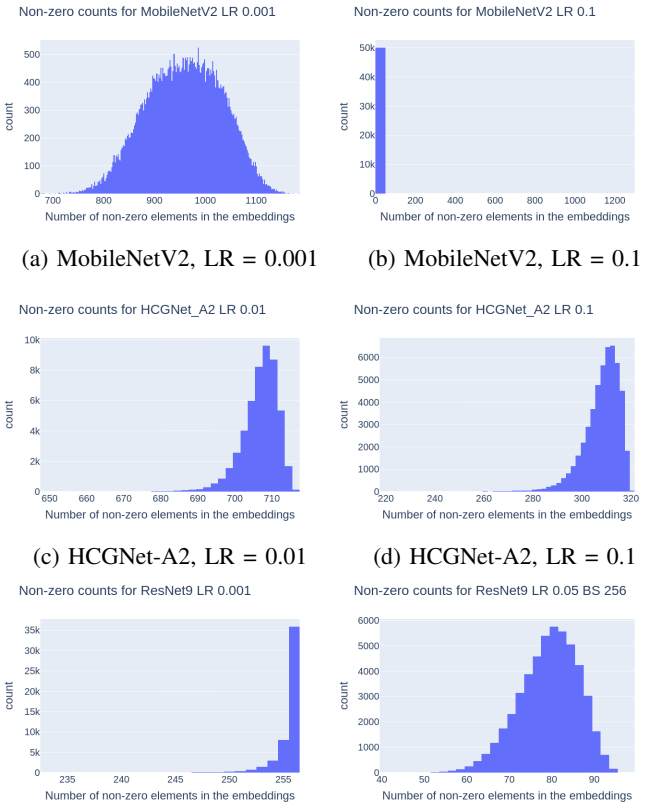
Table II further isolates the effect of batch-size scheduling. When trained with a high learning rate, static batches show reduced accuracy despite strong fault masking. Empirically, the co-scheduling of η and B restores accuracy, while improving sparsity and fault masking compared to fixed-batch training.

B. Analytical View of Sparsity-Induced Fault Masking

A cross-architecture regression over all configurations reveals a clear linear relationship between activation sparsity s and the proportion of masked faults M. The fitted model,

$$M(s) \approx 33.1 + 0.58\, s, \tag{7}$$

achieves $R^2 = 0.83$ and a Pearson correlation coefficient $r = 0.91$. This strong linear dependence demonstrates that sparsity is a dominant, architecture-independent predictor of fault resilience. On average, every 10% increase in sparsity yields approximately a 6% improvement in fault masking. The nonzero intercept ($M_0 \approx 33\%$) indicates that even dense networks exhibit inherent fault-masking capability, likely due to intrinsic redundancy in convolutional layers (also observed in literature [28], [37]).



(a) MobileNetV2, LR = 0.001 (b) MobileNetV2, LR = 0.1

(c) HCGNet-A2, LR = 0.01 (d) HCGNet-A2, LR = 0.1

(e) ResNet9, LR = 0.001 (f) ResNet9, LR = 0.05, BS

Fig. 3: Panels (a)–(f) show distributions of non-zero activations for increasing learning rates. Higher learning rates yield progressively sparser feature embeddings, suppressing non-relevant feature channels. This confirms that training dynamics can implicitly drive activation sparsity, yielding sparse activation patterns correlated with higher fault masking.

Figure 3 illustrates how increasing the learning rate induces sparsity in the feature embeddings. The distribution of non-zero activations shifts sharply toward zero, confirming the implicit regularization effect that underpins the observed robustness trend.

VI. Conclusions and Limitations

This study demonstrated that fault resilience in CNNs can be tuned through training dynamics rather than architectural

modifications. By framing the process via the noise-scale formulation of SGD [35], we showed that learning-rate and batch-size co-scheduling controls the level of sparsity and, consequently, the network's inherent ability to mask faults. The proposed *anti-annealing* regime maintains sufficient gradient noise to enhance robustness while limiting accuracy degradation. Traditional reliability research has relied on post-training redundancy or FAT [7], [9], [20]. In contrast, our approach introduces no explicit fault model, fault injection, or hardware changes. Reliability instead emerges naturally from the optimization process: by regulating sparsity through learning dynamics, the network does not change its architecture but converges toward a sparse feature representation of the input data. This representation-level sparsity, rather than parameter pruning, enables the model to self-organize into a compact yet fault-tolerant configuration. Robustness stems from the structure of feature representations rather than parameter pruning. Our results should be interpreted as evidence of representation-level algorithmic masking. This work opens the door to dynamic reliability modeling, where masking probability evolves with training dynamics. While results are consistent across multiple CNNs and datasets, further evaluation on quantized and mixed-precision models would strengthen generality. Future work will extend the analysis to microarchitectural fault simulations [34] and explore theoretical models for optimal co-scheduling of (η, B).

ACKNOWLEDGMENT

This work has been funded by the ANR France 2030 AdaptING project "ANR-23-PEIA-0009".

REFERENCES

[1] A. Ruospo *et al.*, "A survey on deep learning resilience assessment methodologies," *Computer*, 2023.

[2] S. Mittal, "A survey on modeling and improving reliability of dnn algorithms and accelerators," *J. Syst. Archit.*, Mar. 2020.

[3] N. Bellarmino *et al.*, "Covid-19 detection from exhaled breath," *Scientific Reports*, 2024.

[4] M. A. Neggaz *et al.*, "A reliability study on cnns for critical embedded systems," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018.

[5] E. Malekzadeh *et al.*, "The impact of faults on dnns: A case study," in *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2021.

[6] A. Bosio *et al.*, "A reliability analysis of a deep neural network," in *2019 IEEE Latin American Test Symposium (LATS)*, 2019.

[7] F. Libano *et al.*, "Selective hardening for neural networks in fpgas," *IEEE Transactions on Nuclear Science*, 2019.

[8] U. Zahid *et al.*, "Fat: Training neural networks for reliable inference under hardware faults," in *2020 IEEE International Test Conference (ITC)*, 2020.

[9] N. Cavagnero *et al.*, "Transient-fault-aware design and training to enhance dnns reliability with zero-overhead," in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2022.

[10] M. A. Hanif *et al.*, *Efat: Improving the effectiveness of fault-aware training for mitigating permanent faults in dnn hardware accelerators*, 2023.

[11] Y.-h. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, 2016.

[12] A. Parashar *et al.*, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[13] G. D. Natale *et al.*, *Cross-Layer Reliability of Computing Systems*. Jan. 2020.

[14] S. Pappalardo *et al.*, "A fault injection framework for ai hardware accelerators," in *2023 IEEE 24th Latin American Test Symposium (LATS)*, 2023.

[15] G. Li *et al.*, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017.

[16] N. Bellarmino *et al.*, "Dear-cnn: Data-efficient assessment of resiliency in convolutional neural networks," in *2025 IEEE 28th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2025.

[17] N. Bellarmino *et al.*, "Opra: Optimizing resiliency assessment for deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

[18] N. Cherezova *et al.*, "Can model-level fault tolerance be enough for dnn hardware accelerators? a study," *IEEE Design & Test*, Jan. 2026.

[19] M. H. Ahmadilivani *et al.*, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Computing Surveys*, 2024.

[20] G. Gambardella *et al.*, "Efficient error-tolerant quantized neural network accelerators," in *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, IEEE, Oct. 2019.

[21] Z. Liu *et al.*, "An efficient structure to improve the reliability of deep neural networks on arms," *Microelectronics Reliability*, 2022.

[22] L. Bottou *et al.*, "Optimization Methods for Large-Scale Machine Learning," *Society for Industrial and Applied Mathematics (SIAM) Review*, 2018.

[23] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*, Springer, 2012.

[24] D. G. T. Barrett *et al.*, *Implicit gradient regularization*, 2022.

[25] M. Andriushchenko *et al.*, *Sgd with large step sizes learns sparse features*, 2023.

[26] A. Yaguchi *et al.*, *Adam induces implicit weight sparsity in rectifier neural networks*, 2018.

[27] T. Ganguli *et al.*, "Activation-based pruning of neural networks," *Algorithms*, 2024.

[28] N. Bellarmino *et al.*, "Investigating on Gradient Regularization for Testing Neural Networks," in *LOD 2024 : 10th International Conference on machine Learning, Optimization and Data science*, Riva de Sole, Italy, Sep. 2024.

[29] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[30] A. G. Howard *et al.*, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, 2017.

[31] M. Sandler *et al.*, *Mobilenetv2: Inverted residuals and linear bottlenecks*, 2019.

[32] C. Yang *et al.*, *Gated convolutional networks with hybrid connectivity for image classification*, https://github.com/winycg/HCGNet, 2020.

[33] A. Krizhevsky, *Cifar-10 dataset*, https://www.cs.toronto.edu/~kriz/cifar.html, Accessed on April 11, 2024, 2009.

[34] O. Chatzopoulos *et al.*, "Accurate analysis of silent data corruptions in programmable ai accelerator microarchitectures," in *IEEE Int. Symp. on On-Line Testing and Robust System Design (IOLTS)*, 2025.

[35] S. L. Smith *et al.*, "A bayesian perspective on generalization and stochastic gradient descent," *CoRR*, 2017.

[36] S. L. Smith *et al.*, *Don't decay the learning rate, increase the batch size*, 2018.

[37] S. Hong *et al.*, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *Proceedings of the 28th USENIX Conference on Security Symposium*, ser. SEC'19, Santa Clara, CA, USA: USENIX Association, 2019.