Learning Recoverable Neural Networks against Weight Corruption via Simple Zero-Sum Projection

Sanggeon Yun
Dept. of Computer Science
UC Irvine
Irvine, CA 92697
sanggeoy@uci.edu

Ryozo Masukawa
Dept. of Computer Science
UC Irvine
Irvine, CA 92697
rmasukaw@uci.edu

SungHeon Jeong
Dept. of Computer Science
UC Irvine
Irvine, CA 92697
sungheoj@uci.edu

Hyunwoo Oh
Dept. of Computer Science
UC Irvine
Irvine, CA 92697
hyunwooo@uci.edu

Nathaniel D. Bastian
Institute for Artificial Intelligence
Syracuse University
Syracuse, NY 13244
ndbastia@syr.edu

Mohsen Imani
Dept. of Computer Science
UC Irvine
Irvine, CA 92697
m.imani@uci.edu

Abstract

Deep neural networks are highly vulnerable to bit corruption in stored weights, whether from naturally occurring memory faults or deliberate fault-injection attacks. Existing defenses remain fundamentally limited: fault-tolerance methods degrade under cumulative corruption and remain vulnerable to strong white-box attacks, while ECC-based recovery methods are often tied to specific precisions or architectures and expose concentrated vulnerable surfaces. We propose a simple training-based recovery framework that endows model weights with recoverable structure through block-wise zero-sum constraints enforced by differentiable projection. The resulting method incurs no additional weight-storage overhead and generalizes naturally across architectures and numerical precisions. We further develop adaptive adversarial bit-flip attacks tailored to recovery-based defenses, covering both our method and prior ECC-based approaches. Across diverse architectures, precisions, datasets, threat models, and tasks, our method consistently delivers strong robustness and recoverability while preserving clean performance.

1 Introduction

Deep neural networks (DNNs) have become a central component of modern machine learning systems and are now deployed across a broad range of applications, including computer vision [37, 11], natural language processing [54, 1], and information retrieval [59, 51], and on-device edge AI [31]. As these models increasingly move from cloud infrastructure to resource-constrained and safety-critical platforms, their reliability under low-level hardware faults has emerged as an important challenge. In particular, recent studies have shown that DNNs are highly vulnerable to *bit flips* in stored model weights: even a small number of corrupted bits can cause severe degradation in predictive performance or induce highly undesirable model behaviors [26, 32, 21, 23].

Existing work studies this vulnerability under two broad settings: *random bit flips* and *adversarial bit flips*. Random bit flips capture naturally occurring memory faults caused by low-voltage operation, temperature variation, space radiation, and other hardware-level factors that commonly arise in real-world edge deployment [6, 23, 21, 27, 7, 14, 60, 61]. In such environments, even rare faults can accumulate over time and eventually cause catastrophic accuracy degradation. Adversarial bit flips, in contrast, assume an attacker who intentionally corrupts selected bits in memory to manipulate model

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

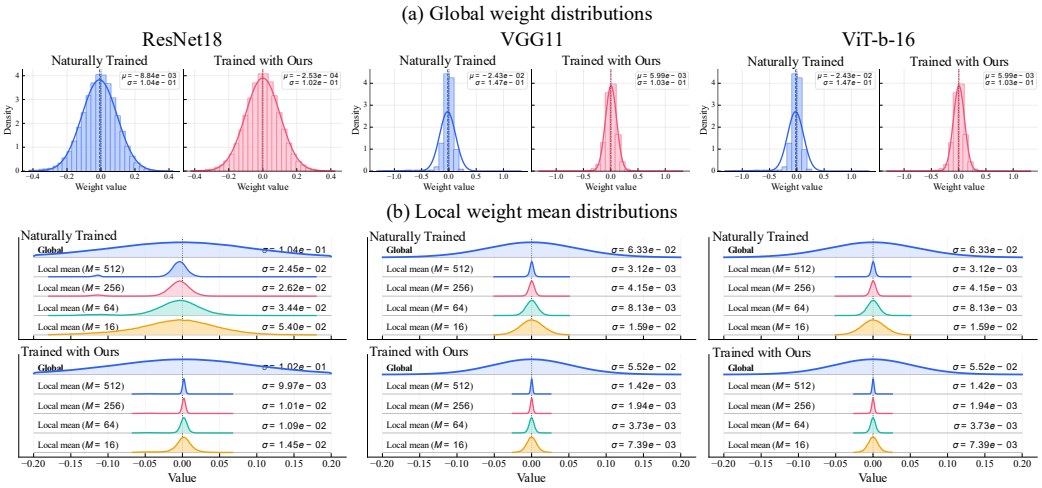

Figure 1: Global and local weight distributions on ImageNet1K for naturally trained models and our method with block size $K = 8$. (a) Across architectures, naturally trained weights are already approximately symmetric and centered near zero; our method preserves this structure while enforcing a more strictly zero-centered distribution. (b) Each local mean is computed over M consecutive flattened weights. Even naturally trained models exhibit near-zero local means across scales, while our method makes this local zero-centering substantially tighter.

behavior [32]. Such attacks may be realized through physical fault-injection mechanisms, including DRAM disturbance attacks such as RowHammer and laser-based fault injection [5, 24, 46, 2]. Because inducing targeted faults is costly and time-consuming even for a small number of bits, a large body of recent work has focused on *bit-efficient* attacks that use only a few or a few tens of bit flips, and identify a small set of highly influential bits whose corruption is sufficient to induce misclassification or backdoor-like behavior [39, 40, 8, 4, 52, 62, 3, 30].

To address these threats without relying on exhaustive redundancy such as replicating multiple copies of model parameters in memory [20], which is especially costly in edge deployment, a range of defenses that avoid additional storage overhead has been proposed. Existing defenses can be broadly divided into *fault tolerance* and *fault recovery*. Fault-tolerance methods, including bit-flip-aware training [49, 10, 50] and output code matching (OCM)-based methods [58, 35], aim to train models whose predictions are less sensitive to weight corruption. Fault-recovery methods include ECC-style protection schemes [36, 16, 28], which embed error-correcting structure into model parameters to detect and recover corrupted bits without additional storage overhead.

Despite their promise, existing defenses face several important practical limitations. First, **existing fault-tolerance methods are fundamentally limited under cumulative errors.** Since these methods do not restore corrupted parameters, we find that their performance degrades progressively as random faults accumulate, while remaining vulnerable to strong adversarial bit-flip attacks. Second, **existing fault-recovery methods often introduce concentrated failure points and can be broken by stronger adaptive adversarial attacks.** Although ECC-based recovery schemes can improve robustness to cumulative random faults through error correction, our experiments show that their recovery capability can be neutralized by only a few carefully selected bit flips under adaptive attacks. This weakness stems from an inherent limitation of ECC itself: for the evaluated SECDED-based scheme, two errors within one codeword already exceed its single-error correction capability. Third, **most existing methods are neither model-agnostic nor precision-agnostic.** Many prior defenses are specialized to particular numerical precisions or architectural families; for instance, most bit-flip-aware training and ECC-based recovery methods are designed mainly for quantized models and relatively narrow architecture classes such as CNNs. Even methods that are, in principle, applicable to non-quantized models, such as OCM, remain ineffective in floating-point settings, where flips in exponent bits can induce drastic numerical changes. In practice, they are only effective in quantized models, which already possess a level of robustness. These limitations motivate the need for a recovery-oriented defense that is robust to both cumulative random faults and adaptive adversarial attacks, while remaining broadly applicable across architectures and numeric precisions.

Inspired by biological findings that *homeostatic plasticity*, which regulates neural activity toward a stable operating range, is a key factor behind the remarkable recovery capacity of the brain [53, 42, 44],

we investigate the weight distributions of naturally trained DNNs across architectures and datasets and observe that they already exhibit a regularized, zero-centered, and approximately symmetric structure as illustrated in Figure 1. Motivated by this observation, **we propose a new training-based fault-recovery method that explicitly enforces this structure through simple zero-sum constraints.** By strengthening a property that DNN weights already naturally tend to exhibit, our method provides model-, precision-, and task-agnostic recoverability without inference time and weight storage overhead. Unlike prior work, which is typically restricted to narrow settings in architecture, precision, threat model, or task, our evaluation covers both random and adversarial bit-flip settings as well as a new adaptive threat model in which the attacker is aware of the recovery mechanism of both our method and prior ECC-based defenses.

2 Threat Models

We consider both standard bit-flip threats studied in prior work, namely random and adversarial bit flips, and additionally propose adaptive adversarial threats that explicitly account for the deployed recovery mechanism, a setting that has not been considered in existing work.

Random bit-flip threats. We follow the standard random-fault model used in prior work, where memory errors are modeled as independent bit corruptions with a prescribed bit error rate (BER) [16, 10, 50, 28]. Concretely, each stored bit in the deployed model is flipped independently according to a Bernoulli distribution with parameter ρ, where ρ denotes the BER. This setting captures the cumulative effect of naturally occurring memory faults and allows controlled evaluation across a wide range of corruption severities.

Adversarial bit-flip threats. We also consider an active adversary who can choose which bits to flip in order to maximize damage under a strict bit budget. Following the existing literature, we evaluate both *destructive* and *stealthy* attacks. Destructive attacks aim to directly collapse benign task performance, with representative examples including BFA-PBS [39]. Stealthy attacks instead aim to preserve benign accuracy while inducing attacker-controlled behavior, typically only when a trigger is present in the input; representative examples include TBT [40], ProFlip [8], HPT [4], TrojViT [62], Deep-TROJ [3], and SoleFlip [30].

Adaptive adversarial bit-flip threats. We further consider *adaptive* attackers that are aware of the deployed recovery mechanism and choose bit flips to evade detection, or destroy recoverability. This setting is particularly important for fault-recovery defenses, since the attacker can exploit the structure of the recovery rule itself, yet it has not been studied in prior work under bit-flip threats. We introduce adaptive attacks for both ECC-based defense and our method in Section 4.

3 Proposed Recovery Method: Simple Zero-Sum Projection

Algorithm 1: Training with simple zero-sum projection

Input: training data $\{(x_i, y_i)\}$, block size K, trainable tensors $\{A^{(\ell)}\}_{\ell=1}^{L}$

for *each training iteration* **do**
  for $\ell = 1, \ldots, L$ **do**
    $\big(B_1^{(\ell)}, \ldots, B_{M_\ell}^{(\ell)}\big) \leftarrow G_K(A^{(\ell)})$;
    $W^{(\ell)} \leftarrow U_K\big(\Pi(B_1^{(\ell)}), \ldots, \Pi(B_{M_\ell}^{(\ell)})\big)$;
    map $W^{(\ell)}$ to the deployed precision ; `// e.g., fixed-scale quantization or FP cast`
  update $\{A^{(\ell)}\}_{\ell=1}^{L}$ using gradients through $f(x; \Theta)$;

Algorithm 2: Block detection and recovery

Input: corrupted block $Y \in \mathbb{R}^{q \times q}$, threshold ε

$r \leftarrow Y\mathbf{1}_q,\ c \leftarrow Y^\top \mathbf{1}_q$;
if $\|r\|_\infty \le \varepsilon$ ***and*** $\|c\|_\infty \le \varepsilon$ ***and*** *finite* **then**
  return Y
$I \leftarrow \{i : |r_i| > \varepsilon\},\ J \leftarrow \{j : |c_j| > \varepsilon\},\ \mathcal{N} \leftarrow \{(i,j) : Y_{ij} \text{ is non-finite}\}\ D \leftarrow (I \times J) \cup \mathcal{N}$;
while *some row/column isolates a unique* $(i,j) \in D$ **do**
  $\widehat{E}_{ij} \leftarrow r_i$ or $c_j, \quad Y \leftarrow Y - \widehat{E}$;
  update r, c, D;
construct S^{row} and S^{col} from the residuals r, c over D;
$Y \leftarrow Y - \frac{1}{2}(S^{\mathrm{row}} + S^{\mathrm{col}})$; `// soft correction on remaining` D
return Y

Motivation. Biological neural systems maintain stable activity after localized damage through self-stabilizing mechanisms, with *homeostatic plasticity* often viewed as a central principle that regulates neural activity toward a stable operating range [53, 42, 44]. Inspired by this perspective, we seek a recovery mechanism that is learned *within* the weight space rather than injected *onto* it through external check bits. As shown in Figure 1, empirically, naturally trained DNN weights are already approximately zero-centered, suggesting that constraining them to a structured zero-sum subspace is substantially less disruptive than replacing part of the representation with ECC payloads. This motivates our block-wise zero-sum design, which preserves clean accuracy while endowing deployed weights with observable and correctable structure.

Block-wise zero-sum constraint. Consider a DNN $f(x;\Theta)$, where x is the input and $\Theta = \{W^{(1)}, \dots, W^{(L)}\}$ denotes the deployed weight tensors of its L layers. For layer ℓ, let $W^{(\ell)} \in \mathbb{R}^{d_1^{(\ell)} \times \cdots \times d_{m_\ell}^{(\ell)}}$ be its deployed weight tensor. Instead of directly optimizing $W^{(\ell)}$, we introduce a trainable tensor $A^{(\ell)}$ of the same shape and define

$$W^{(\ell)} = \Phi\left(A^{(\ell)}\right), \tag{1}$$

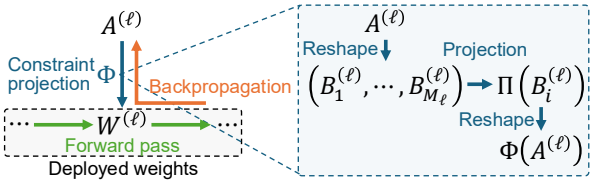

Figure 2: Training with zero-sum projection. Trainable weights $A^{(\ell)}$ are partitioned into blocks, projected, and reassembled into $W^{(\ell)} = \Phi(A^{(\ell)})$, with gradients propagated through Φ.

where Φ transforms each weight tensor into one that satisfies local zero-sum constraints. This transformation is defined with respect to a block size K. Specifically, let $\mathcal{G}_K(\cdot)$ denote the operation that partitions a tensor into consecutive square blocks under the leftover-merge rule, and let $\mathcal{U}_K(\cdot)$ denote the inverse operation that assembles the processed blocks back into the original tensor shape. Applying $\mathcal{G}_K$ to $A^{(\ell)}$ yields

$$\mathcal{G}_K(A^{(\ell)}) = \left(B_1^{(\ell)}, \dots, B_{M_\ell}^{(\ell)}\right), \tag{2}$$

where each regular block satisfies $B_t^{(\ell)} \in \mathbb{R}^{K \times K}$, and there is at most one final tail block. When a tail block is present, it is formed by merging the remainder with the previous block and embedding the valid entries into a $(K+1) \times (K+1)$ square. The deployed tensor is then defined by

$$\Phi(A^{(\ell)}) = \mathcal{U}_K\big(\Pi(B_1^{(\ell)}), \dots, \Pi(B_{M_\ell}^{(\ell)})\big), \tag{3}$$

where $\Pi(\cdot)$ denotes the zero-sum map applied blockwise.

For a regular block $B \in \mathbb{R}^{K \times K}$, the target constraint is

$$B\mathbf{1}_K = \mathbf{0}, \qquad \mathbf{1}_K^\top B = \mathbf{0}^\top, \tag{4}$$

where $\mathbf{1}_K \in \mathbb{R}^K$ is the all-ones vector. Thus each regular block lies in the zero-sum subspace

$$\mathcal{S}_K = \left\{B \in \mathbb{R}^{K \times K} : \; B\mathbf{1}_K = \mathbf{0}, \; \mathbf{1}_K^\top B = \mathbf{0}^\top\right\}. \tag{5}$$

Unbalanced zero-sum projection baseline. A direct but asymmetric way to enforce the zero-sum constraint is to parameterize each block through its free coordinates. For a square block of side length q, let $\widetilde{B} \in \mathbb{R}^{(q-1)\times(q-1)}$ and define

$$\Pi_{\text{unbalanced}}(\widetilde{B}) = \begin{bmatrix} \widetilde{B} & -\widetilde{B}\mathbf{1}_{q-1} \\ -\mathbf{1}_{q-1}^\top \widetilde{B} & \mathbf{1}_{q-1}^\top \widetilde{B}\mathbf{1}_{q-1} \end{bmatrix}. \tag{6}$$

This construction satisfies the zero-sum constraint exactly and shows that a $q \times q$ zero-sum block has only $(q-1)^2$ effective degrees of freedom. We use this as a baseline by setting $\Pi = \Pi_{\text{unbalanced}}$ blockwise. Although exact, this construction assigns a special role to the last row and column of each block and optimizes in a fixed coordinate chart, which can make training less balanced and distort scale allocation, especially under low-precision deployment.

Balanced zero-sum projection. Our main method instead uses a symmetric projection that treats all coordinates uniformly. For any square block $B \in \mathbb{R}^{q\times q}$, define

$$\Pi_{\mathrm{balanced}}(B) = B - \frac{1}{q}B\mathbf{1}_q\mathbf{1}_q^\top - \frac{1}{q}\mathbf{1}_q\mathbf{1}_q^\top B + \frac{1}{q^2}\mathbf{1}_q\mathbf{1}_q^\top B\mathbf{1}_q\mathbf{1}_q^\top. \tag{7}$$

We set $\Pi = \Pi_{\mathrm{balanced}}$ blockwise. Compared with $\Pi_{\mathrm{unbalanced}}$, this balanced form preserves the original parameter geometry more naturally, avoids privileging specific coordinates, and remains fully differentiable through standard tensor operations.

Deployment across precisions. After zero-sum projection, the constrained tensor is mapped to the deployed numerical precision. For floating-point deployment, we use $W_{\mathrm{fp32}}^{(\ell)} = \Phi(A^{(\ell)})$ or $W_{\mathrm{fp16}}^{(\ell)} = \mathrm{cast}_{16}(\Phi(A^{(\ell)}))$, with a straight-through cast during training for FP16 deployment. For signed symmetric b-bit post-quantization, we use

$$Q^{(\ell)} = \mathrm{clip}\left(\left\lfloor \frac{\Phi(A^{(\ell)})}{\Delta^{(\ell)}} \right\rceil, -q_{\max}, q_{\max}\right), \qquad W_{\mathrm{q}}^{(\ell)} = \Delta^{(\ell)}Q^{(\ell)}, \tag{8}$$

where $q_{\max} = 2^{b-1} - 1$ and $\Delta^{(\ell)}$ is a fixed layer scale. Rather than deriving $\Delta^{(\ell)}$ from the realized layer maximum, we set it as a constant significance-level hyperparameter based on the target normal-distribution range, so that the scale is a single, easily protected fixed configuration constant rather than mutable layer metadata that introduces an additional attack surface (with negligible $O(1)$ protection storage, compared with $O(L)$ for mutable scales across L layers). This preserves the intended weight distribution while removing the usual attack surface associated with mutable per-layer scales. During training, we apply Equation 8 via quantization-aware training, so the recoverable structure is learned directly under the target deployed precision.

Detection and recovery. For a clean zero-sum block $X \in \mathbb{R}^{q\times q}$, where $q = K$ for a regular block and $q = K + 1$ for the unique tail block, define the row and column residual vectors

$$r(X) = X\mathbf{1}_q, \qquad c(X) = X^\top\mathbf{1}_q, \tag{9}$$

where $\mathbf{1}_q \in \mathbb{R}^q$ is the all-ones vector. By construction, $r(X) = \mathbf{0}$ and $c(X) = \mathbf{0}$. After corruption, we observe

$$Y = X + E, \tag{10}$$

where $E \in \mathbb{R}^{q\times q}$ is an unknown error pattern. We detect a block as corrupted whenever

$$\|r(Y)\|_\infty > \varepsilon \quad \text{or} \quad \|c(Y)\|_\infty > \varepsilon, \tag{11}$$

or when non-finite entries appear. Here $\varepsilon \geq 0$ is a recovery threshold hyperparameter chosen from the clean deployed model, e.g., as the maximum clean residual magnitude over all blocks, so that numerical imprecision does not trigger false positives and inadvertently damage an uncorrupted model during recovery.

Let

$$\mathcal{I} = \{i \in \{1,\dots,q\} : |r_i(Y)| > \varepsilon\}, \qquad \mathcal{J} = \{j \in \{1,\dots,q\} : |c_j(Y)| > \varepsilon\}, \tag{12}$$

and define the candidate support

$$\mathcal{D} = (\mathcal{I}\times\mathcal{J}) \cup \mathcal{N}, \tag{13}$$

where $\mathcal{N} \subseteq \{1,\dots,q\}\times\{1,\dots,q\}$ is the set of non-finite locations. Recovery first applies exact updates whenever a row or column isolates a unique candidate in $\mathcal{D}$. If $(i,j) \in \mathcal{D}$ is uniquely isolated, we set

$$\widehat{E}_{ij} = \begin{cases} r_i(Y), & \text{if row } i \text{ has a unique candidate,} \\ c_j(Y), & \text{if column } j \text{ has a unique candidate,} \end{cases} \qquad Y \leftarrow Y - \widehat{E}. \tag{14}$$

After no further exact updates are available, the remaining residual is distributed over $\mathcal{D}$ by

$$Y \leftarrow Y - \frac{1}{2}\big(S^{\mathrm{row}} + S^{\mathrm{col}}\big), \tag{15}$$

where $S^{\mathrm{row}}, S^{\mathrm{col}} \in \mathbb{R}^{q\times q}$ allocate the current row and column residuals uniformly over the remaining candidate entries in each row and column. For the tail block, the same procedure is applied on the padded $(K+1)\times(K+1)$ block and only the valid entries are retained afterward. All steps use only sums, masks, broadcasts, and batched updates, so detection and recovery remain fully tensorized.

Further theoretical analysis. A formal theoretical analysis, including exact recovery guarantees, probabilistic robustness under random corruption, and comparison with ECC-based recovery, is deferred to Appendix I.

4 Proposed Adaptive Bit-Flip Attacks

Standard adversarial bit-flip attacks optimize attack success on the corrupted model itself, without taking the deployed recovery rule into account. For recovery-based defenses, we therefore consider a stronger adaptive attacker that evaluates flips by their *post-recovery* effect. Concretely, the attacker first uses the underlying attack method to identify a set of promising bit-flip candidates. It then applies the defense recovery procedure to each candidate-corrupted model and scores that candidate according to the resulting post-recovery attack objective. In this way, candidate selection remains guided by the original attack mechanism, while the final bit choice is determined by how effective the flip remains after recovery. This procedure naturally extends to recovery rules that are discrete or non-differentiable, such as ECC-based methods, since recovery is used only to evaluate candidate flips, rather than through end-to-end gradient propagation.

Adaptive destructive attack. For destructive attacks, the attacker selects bit flips to maximize the attack objective *after recovery* rather than before recovery. Let W denote the deployed weights, ΔW the bit-flip perturbation constrained to a feasible set $\mathcal{B}$, $\mathcal{R}(\cdot)$ the recovery operator, and $\mathcal{L}_{\mathrm{atk}}$ the destructive attack objective. The adaptive attacker targets

$$\max_{\Delta W \in \mathcal{B}} \; \mathcal{L}_{\mathrm{atk}}(\mathcal{R}(W + \Delta W)) \,. \tag{16}$$

and greedily selects flips according to their post-recovery effect. Thus, each bit is chosen based on the damage it causes after the defense has applied recovery, directly targeting the recoverability of the defense rather than only the vulnerability of the unrecovered model.

Adaptive stealthy attack. For stealthy attacks, we use the same principle, but with the original stealthy attack objective. Let $\mathcal{L}_{\mathrm{stealth}}$ denote the objective of the underlying stealthy attack, which promotes targeted behavior on triggered inputs while penalizing degradation on clean inputs. The adaptive attacker instead targets

$$\max_{\Delta W \in \mathcal{B}} \; \mathcal{L}_{\mathrm{stealth}}(\mathcal{R}(W + \Delta W)) \,. \tag{17}$$

so that selected flips remain effective even after the recovery mechanism is executed. In this way, the attacker jointly optimizes stealth and post-recovery attack efficacy, yielding a substantially stronger threat model for recovery-based defenses even when the recovery rule itself is not gradient-flowable.

5 Experiments

5.1 Experimental Setup

We conduct experiments on a wide range of datasets, architectures, numerical precisions, and threat models to evaluate the robustness and generality of the proposed method. The main experiments are run on an NVIDIA H200 GPU with 150 GB memory.

Datasets. We evaluate on five benchmark datasets that cover a broad range of scales and visual domains: CIFAR-10, CIFAR-100 [25], GTSRB [48], ImageNet-1K [45], and SVHN [34].

Models and precisions. To assess architectural generality, we consider both CNN-based and Transformer-based models, including ResNet-18, ResNet-34, VGG-11, VGG-13, and ViT-B/16 [17, 47, 13]. All models are trained for 100 epochs. We evaluate both FLOAT32 and INT8 settings. For INT8 experiments, we use quantization-aware training. For our method, INT8 models use a fixed quantization scale $\Delta = a/127$, corresponding to a symmetric clipping range $[-a, a]$ with $a = 2.576$. Additional LSTM [15] and U-Net [43] evaluations are reported in Appendix F to further demonstrate the task-agnostic applicability of our method.

Table 1: Cumulative random bit-flip robustness in FLOAT32 at a BER of $10^{-6}\%$. Values report the mean $\pm$ standard deviation of robust accuracy over three runs. The best result in each setting is **highlighted**.

Dataset	Method	Resnet-34					VGG-13				
		1 step	2 steps	4 steps	8 steps	10 steps	1 steps	2 steps	4 steps	8 steps	10 steps
CIFAR10	None	$0.3996_{\pm 0.4480}$	$0.3997_{\pm 0.4483}$	$0.1021_{\pm 0.0114}$	$0.0957_{\pm 0.0007}$	$0.0957_{\pm 0.0007}$	$0.7998_{\pm 0.0532}$	$0.7217_{\pm 0.0674}$	$0.3971_{\pm 0.3499}$	$0.2495_{\pm 0.2673}$	$0.0993_{\pm 0.0125}$
	OCM	$0.6452_{\pm 0.4763}$	$0.6450_{\pm 0.4761}$	$0.1013_{\pm 0.0086}$	$0.0969_{\pm 0.0024}$	$0.0952_{\pm 0.0000}$	$0.8356_{\pm 0.0701}$	$0.7594_{\pm 0.0503}$	$0.4378_{\pm 0.3045}$	$0.1328_{\pm 0.0506}$	$0.0950_{\pm 0.0003}$
	ECOC	$0.3698_{\pm 0.4722}$	$0.1014_{\pm 0.0074}$	$0.0723_{\pm 0.0535}$	$0.1038_{\pm 0.0079}$	$0.0981_{\pm 0.0000}$	$0.7030_{\pm 0.0367}$	$0.6577_{\pm 0.0373}$	$0.3654_{\pm 0.2141}$	$0.1048_{\pm 0.0070}$	$0.1004_{\pm 0.0031}$
	Ours-U	$0.9028_{\pm 0.0000}$	$0.9028_{\pm 0.0000}$	$0.9028_{\pm 0.0000}$	$0.9028_{\pm 0.0000}$	$0.9028_{\pm 0.0000}$	$0.9106_{\pm 0.0000}$	$0.9106_{\pm 0.0000}$	$0.9109_{\pm 0.0003}$	$0.9109_{\pm 0.0003}$	$0.9109_{\pm 0.0003}$
	Ours-B	$\mathbf{0.9062}_{\pm 0.0000}$	$\mathbf{0.9062}_{\pm 0.0000}$	$\mathbf{0.9062}_{\pm 0.0000}$	$\mathbf{0.9060}_{\pm 0.0003}$	$\mathbf{0.9062}_{\pm 0.0000}$	$\mathbf{0.9189}_{\pm 0.0000}$	$\mathbf{0.9189}_{\pm 0.0000}$	$\mathbf{0.9189}_{\pm 0.0000}$	$\mathbf{0.9189}_{\pm 0.0000}$	$\mathbf{0.9189}_{\pm 0.0000}$
CIFAR100	None	$0.4907_{\pm 0.3539}$	$0.2663_{\pm 0.3727}$	$0.2609_{\pm 0.3779}$	$0.0106_{\pm 0.0020}$	$0.0107_{\pm 0.0028}$	$0.6102_{\pm 0.0119}$	$0.5443_{\pm 0.0555}$	$0.2879_{\pm 0.2384}$	$0.1377_{\pm 0.2203}$	$0.0090_{\pm 0.0010}$
	OCM	$0.2404_{\pm 0.4024}$	$0.2407_{\pm 0.4030}$	$0.0243_{\pm 0.0281}$	$0.0099_{\pm 0.0012}$	$0.0093_{\pm 0.0007}$	$0.5184_{\pm 0.0861}$	$0.4705_{\pm 0.0792}$	$0.3081_{\pm 0.2587}$	$0.1222_{\pm 0.1777}$	$0.0106_{\pm 0.0025}$
	ECOC	$0.2450_{\pm 0.4052}$	$0.0111_{\pm 0.0017}$	$0.0085_{\pm 0.0033}$	$0.0093_{\pm 0.0000}$	$0.0093_{\pm 0.0000}$	$0.5851_{\pm 0.0428}$	$0.5238_{\pm 0.0653}$	$0.3171_{\pm 0.2676}$	$0.1382_{\pm 0.2233}$	$0.0112_{\pm 0.0035}$
	Ours-U	$0.6841_{\pm 0.0005}$	$0.6843_{\pm 0.0003}$	$0.6841_{\pm 0.0005}$	$0.6841_{\pm 0.0000}$	$0.6841_{\pm 0.0005}$	$0.6812_{\pm 0.0000}$	$0.6812_{\pm 0.0000}$	$0.6814_{\pm 0.0003}$	$0.6812_{\pm 0.0000}$	$0.6812_{\pm 0.0000}$
	Ours-B	$\mathbf{0.6982}_{\pm 0.0000}$	$\mathbf{0.6982}_{\pm 0.0000}$	$\mathbf{0.6982}_{\pm 0.0000}$	$\mathbf{0.6982}_{\pm 0.0000}$	$\mathbf{0.6982}_{\pm 0.0000}$	$\mathbf{0.6909}_{\pm 0.0000}$	$\mathbf{0.6909}_{\pm 0.0000}$	$\mathbf{0.6909}_{\pm 0.0000}$	$\mathbf{0.6909}_{\pm 0.0000}$	$\mathbf{0.6909}_{\pm 0.0000}$
GTSRB	None	$0.3534_{\pm 0.5494}$	$0.0270_{\pm 0.0341}$	$0.0262_{\pm 0.0327}$	$0.0034_{\pm 0.0000}$	$0.0034_{\pm 0.0000}$	$0.8073_{\pm 0.2526}$	$0.6196_{\pm 0.3234}$	$0.3548_{\pm 0.2030}$	$0.0609_{\pm 0.0264}$	$0.0578_{\pm 0.0216}$
	OCM	$0.3568_{\pm 0.5485}$	$0.0293_{\pm 0.0274}$	$0.0360_{\pm 0.0268}$	$0.0166_{\pm 0.0186}$	$0.0034_{\pm 0.0000}$	$0.8594_{\pm 0.1187}$	$0.7920_{\pm 0.1125}$	$0.5124_{\pm 0.3423}$	$0.1016_{\pm 0.0473}$	$0.0391_{\pm 0.0504}$
	ECOC	$0.3522_{\pm 0.5538}$	$0.0410_{\pm 0.0150}$	$0.0378_{\pm 0.0200}$	$0.0356_{\pm 0.0000}$	$0.0356_{\pm 0.0000}$	$0.7095_{\pm 0.1109}$	$0.5550_{\pm 0.1838}$	$0.4297_{\pm 0.1804}$	$0.2412_{\pm 0.2200}$	$0.0968_{\pm 0.1064}$
	Ours-U	$0.9741_{\pm 0.0000}$	$0.9741_{\pm 0.0000}$	$0.9741_{\pm 0.0000}$	$0.9741_{\pm 0.0000}$	$0.9741_{\pm 0.0000}$	$\mathbf{0.9814}_{\pm 0.0000}$	$0.9812_{\pm 0.0003}$	$0.9812_{\pm 0.0003}$	$0.9812_{\pm 0.0003}$	$0.9812_{\pm 0.0003}$
	Ours-B	$\mathbf{0.9771}_{\pm 0.0000}$	$\mathbf{0.9771}_{\pm 0.0000}$	$\mathbf{0.9771}_{\pm 0.0000}$	$\mathbf{0.9771}_{\pm 0.0000}$	$\mathbf{0.9771}_{\pm 0.0000}$	$\mathbf{0.9814}_{\pm 0.0000}$	$\mathbf{0.9814}_{\pm 0.0000}$	$\mathbf{0.9814}_{\pm 0.0000}$	$\mathbf{0.9814}_{\pm 0.0000}$	$\mathbf{0.9814}_{\pm 0.0000}$
ImageNet1K	None	$0.3972_{\pm 0.4395}$	$0.5000_{\pm 0.3582}$	$0.4712_{\pm 0.4072}$	$0.0256_{\pm 0.0426}$	$0.0256_{\pm 0.0426}$	$0.4149_{\pm 0.2794}$	$0.3402_{\pm 0.2303}$	$0.2388_{\pm 0.1714}$	$0.0877_{\pm 0.1460}$	$0.0032_{\pm 0.0024}$
	OCM	$0.3649_{\pm 0.3148}$	$0.3649_{\pm 0.3148}$	$0.1839_{\pm 0.3169}$	$0.0007_{\pm 0.0010}$	$0.0010_{\pm 0.0007}$	$0.3740_{\pm 0.2698}$	$0.3052_{\pm 0.2186}$	$0.2479_{\pm 0.2030}$	$0.0911_{\pm 0.1532}$	$0.0020_{\pm 0.0018}$
	ECOC	$0.4663_{\pm 0.0038}$	$0.3254_{\pm 0.2425}$	$0.1709_{\pm 0.2572}$	$0.0442_{\pm 0.0618}$	$0.0301_{\pm 0.0500}$	$0.3714_{\pm 0.2620}$	$0.1781_{\pm 0.2456}$	$0.1395_{\pm 0.2205}$	$0.0050_{\pm 0.0010}$	$0.0029_{\pm 0.0013}$
	Ours-U	$0.6592_{\pm 0.0000}$	$0.6592_{\pm 0.0000}$	$0.6592_{\pm 0.0000}$	$0.6594_{\pm 0.0003}$	$0.6602_{\pm 0.0013}$	$0.6230_{\pm 0.0000}$	$0.6225_{\pm 0.0000}$	$0.6225_{\pm 0.0002}$	$0.6225_{\pm 0.0000}$	$0.6225_{\pm 0.0012}$
	Ours-B	$\mathbf{0.7053}_{\pm 0.0003}$	$\mathbf{0.7053}_{\pm 0.0003}$	$\mathbf{0.7051}_{\pm 0.0005}$	$\mathbf{0.7041}_{\pm 0.0014}$	$\mathbf{0.7043}_{\pm 0.0012}$	$\mathbf{0.6450}_{\pm 0.0000}$	$\mathbf{0.6450}_{\pm 0.0000}$	$\mathbf{0.6448}_{\pm 0.0003}$	$\mathbf{0.6445}_{\pm 0.0007}$	$\mathbf{0.6452}_{\pm 0.0012}$
SVHN	None	$0.4287_{\pm 0.4645}$	$0.4289_{\pm 0.4647}$	$0.1619_{\pm 0.0508}$	$0.1306_{\pm 0.0949}$	$0.1306_{\pm 0.0949}$	$0.8657_{\pm 0.0890}$	$0.7765_{\pm 0.1711}$	$0.7261_{\pm 0.1225}$	$0.1937_{\pm 0.1680}$	$0.1196_{\pm 0.0708}$
	OCM	$0.5149_{\pm 0.6384}$	$0.6652_{\pm 0.5211}$	$0.3641_{\pm 0.5215}$	$0.1155_{\pm 0.0735}$	$0.0635_{\pm 0.0000}$	$0.8504_{\pm 0.1831}$	$0.7972_{\pm 0.1776}$	$0.5262_{\pm 0.3619}$	$0.0850_{\pm 0.0210}$	$0.0848_{\pm 0.0203}$
	ECOC	$0.4217_{\pm 0.4709}$	$0.4217_{\pm 0.4709}$	$0.4229_{\pm 0.4699}$	$0.1558_{\pm 0.0369}$	$0.0928_{\pm 0.0314}$	$0.9294_{\pm 0.0256}$	$0.8584_{\pm 0.0861}$	$0.4618_{\pm 0.2665}$	$0.2091_{\pm 0.1035}$	$0.1909_{\pm 0.1087}$
	Ours-U	$0.9590_{\pm 0.0000}$	$0.9590_{\pm 0.0000}$	$0.9590_{\pm 0.0000}$	$0.9592_{\pm 0.0003}$	$0.9592_{\pm 0.0003}$	$\mathbf{0.9648}_{\pm 0.0000}$	$\mathbf{0.9648}_{\pm 0.0000}$	$\mathbf{0.9648}_{\pm 0.0000}$	$\mathbf{0.9648}_{\pm 0.0000}$	$\mathbf{0.9648}_{\pm 0.0000}$
	Ours-B	$\mathbf{0.9658}_{\pm 0.0000}$	$\mathbf{0.9658}_{\pm 0.0000}$	$\mathbf{0.9658}_{\pm 0.0000}$	$\mathbf{0.9658}_{\pm 0.0000}$	$\mathbf{0.9658}_{\pm 0.0000}$	$0.9565_{\pm 0.0000}$	$0.9565_{\pm 0.0000}$	$0.9565_{\pm 0.0000}$	$0.9563_{\pm 0.0003}$	$0.9563_{\pm 0.0003}$

Table 2: Cumulative random bit-flip robustness in FLOAT32 at a BER of $10^{-6}\%$ for ViT-B/16. Values report the mean $\pm$ standard deviation of robust accuracy over three runs. The best result in each setting is **highlighted**.

Dataset	Method	ViT-B/16									
		1 step	2 step	3 step	4 step	5 step	6 step	7 step	8 step	9 step	10 step
CIFAR10	None	$0.4160_{\pm 0.4523}$	$0.0962_{\pm 0.0000}$	$0.0962_{\pm 0.0000}$	$0.0962_{\pm 0.0000}$	$0.0962_{\pm 0.0000}$	$0.0962_{\pm 0.0000}$	$0.0962_{\pm 0.0000}$	$0.0962_{\pm 0.0000}$	$0.0962_{\pm 0.0000}$	$0.0962_{\pm 0.0000}$
	OCM	$0.4065_{\pm 0.4409}$	$0.0947_{\pm 0.0000}$	$0.0947_{\pm 0.0000}$	$0.0947_{\pm 0.0000}$	$0.0947_{\pm 0.0000}$	$0.0947_{\pm 0.0000}$	$0.0947_{\pm 0.0000}$	$0.0947_{\pm 0.0000}$	$0.0947_{\pm 0.0000}$	$0.0947_{\pm 0.0000}$
	ECOC	$0.4050_{\pm 0.4381}$	$0.0952_{\pm 0.0000}$	$0.0952_{\pm 0.0000}$	$0.0952_{\pm 0.0000}$	$0.0952_{\pm 0.0000}$	$0.0952_{\pm 0.0000}$	$0.0952_{\pm 0.0000}$	$0.0952_{\pm 0.0000}$	$0.0952_{\pm 0.0000}$	$0.0952_{\pm 0.0000}$
	Ours-U	$0.7517_{\pm 0.0003}$	$0.7517_{\pm 0.0003}$	$0.7517_{\pm 0.0003}$	$0.7517_{\pm 0.0003}$	$0.7515_{\pm 0.0000}$	$0.7512_{\pm 0.0003}$	$0.7512_{\pm 0.0003}$	$0.7515_{\pm 0.0000}$	$0.7517_{\pm 0.0003}$	$0.7517_{\pm 0.0003}$
	Ours-B	$\mathbf{0.7673}_{\pm 0.0003}$	$\mathbf{0.7673}_{\pm 0.0003}$	$\mathbf{0.7673}_{\pm 0.0003}$	$\mathbf{0.7673}_{\pm 0.0003}$	$\mathbf{0.7673}_{\pm 0.0003}$	$\mathbf{0.7668}_{\pm 0.0003}$	$\mathbf{0.7671}_{\pm 0.0005}$	$\mathbf{0.7671}_{\pm 0.0000}$	$\mathbf{0.7668}_{\pm 0.0003}$	$\mathbf{0.7666}_{\pm 0.0005}$
ImageNet1K	None	$0.3792_{\pm 0.5341}$	$0.0015_{\pm 0.0000}$	$0.0015_{\pm 0.0000}$	$0.0015_{\pm 0.0000}$	$0.0015_{\pm 0.0000}$	$0.0015_{\pm 0.0000}$	$0.0015_{\pm 0.0000}$	$0.0015_{\pm 0.0000}$	$0.0015_{\pm 0.0000}$	$0.0015_{\pm 0.0000}$
	OCM	$0.3115_{\pm 0.4392}$	$0.0010_{\pm 0.0000}$	$0.0010_{\pm 0.0000}$	$0.0010_{\pm 0.0000}$	$0.0010_{\pm 0.0000}$	$0.0010_{\pm 0.0000}$	$0.0010_{\pm 0.0000}$	$0.0010_{\pm 0.0000}$	$0.0010_{\pm 0.0000}$	$0.0010_{\pm 0.0000}$
	ECOC	$0.3289_{\pm 0.4623}$	$0.0020_{\pm 0.0000}$	$0.0020_{\pm 0.0000}$	$0.0020_{\pm 0.0000}$	$0.0020_{\pm 0.0000}$	$0.0020_{\pm 0.0000}$	$0.0020_{\pm 0.0000}$	$0.0020_{\pm 0.0000}$	$0.0020_{\pm 0.0000}$	$0.0020_{\pm 0.0000}$
	Ours-U	$0.4006_{\pm 0.0003}$	$0.4004_{\pm 0.0005}$	$0.4001_{\pm 0.0003}$	$0.4004_{\pm 0.0007}$	$0.4004_{\pm 0.0005}$	$0.4004_{\pm 0.0005}$	$0.4004_{\pm 0.0005}$	$0.4001_{\pm 0.0003}$	$0.3994_{\pm 0.0013}$	$0.3992_{\pm 0.0017}$
	Ours-B	$\mathbf{0.7266}_{\pm 0.0000}$	$\mathbf{0.7266}_{\pm 0.0000}$	$\mathbf{0.7266}_{\pm 0.0000}$	$\mathbf{0.7268}_{\pm 0.0003}$	$\mathbf{0.7266}_{\pm 0.0000}$	$\mathbf{0.7271}_{\pm 0.0007}$	$\mathbf{0.7271}_{\pm 0.0005}$	$\mathbf{0.7271}_{\pm 0.0005}$	$\mathbf{0.7271}_{\pm 0.0007}$	$\mathbf{0.7266}_{\pm 0.0000}$

Baselines. We compare against a broad set of prior defenses, including the bit-flip-aware training method WCAT [10], code-based fault-tolerance methods OCM [35] and ECOC [58], and the ECC-based recovery method ECC [16]. We use WCAT as our primary training-based baseline, as it is a more recently published method in the same broader line of bit-flip-aware robustness training. Additional comparisons with RQUANT+Clipping, RandBET, and AdvBET-FO [49] are provided in Appendix G. We also evaluate two variants of our method: the unbalanced zero-sum projection (**Ours-U**) and the balanced zero-sum projection (**Ours-B**). WCAT and ECC are included only in the quantized setting, and ECC is evaluated only on CNN-style architectures. For ECC, following the original paper, we use an SECDED-style coding scheme embedded into quantized weights, in which each protection block contains 57 payload bits and 7 parity bits, corresponding to 8 protected INT8 values per block. For our method, we use a block size of $K = 8$ unless otherwise specified. For OCM-based methods, we set the code length to 16, 64, 128, and 1024 for datasets with 10, 43, 100, and 1000 classes, respectively.

Adversarial bit-flip attacks. We evaluate against one destructive bit-flip attack, BFA-PBS [39], and six stealthy attacks: TBT [40], ProFlip [8], HPT [4], TrojViT [62], Deep-TROJ [3], and SoleFlip [30]. We allow attacks over the full deployed parameter representation. In FLOAT32, this includes sign, exponent, and mantissa bits. In INT8, this includes all 8 quantized weight bits as well as mutable scale bits. The fixed scale used by Ours-U and Ours-B is treated as protected configuration; Appendix E additionally compares all methods under the same fixed scale to control for scale differences, although this constraint may disadvantage the existing methods that rely on scale adjustment for robustness.

Evaluation metric. Our primary metric is *robust accuracy*. For random bit-flip and destructive attack evaluations, robust accuracy is measured on clean test inputs under corrupted models. For stealthy attacks, robust accuracy is measured on triggered test inputs under corrupted models.

Table 3: Cumulative random bit-flip robustness in INT8 at a BER of $10^{-2}\%$. Values report the mean $\pm$ standard deviation of robust accuracy over three runs. The best result in each setting is **highlighted**.

Dataset	Method	ResNet-34					VGG-13				
		1 step	2 steps	4 steps	8 steps	10 steps	1 steps	2 steps	4 steps	8 steps	10 steps
CIFAR10	None	0.9124$_{\pm 0.0016}$	0.9128$_{\pm 0.0028}$	0.9131$_{\pm 0.0015}$	0.8154$_{\pm 0.1590}$	0.8105$_{\pm 0.1650}$	0.9115$_{\pm 0.0031}$	0.9128$_{\pm 0.0017}$	0.9095$_{\pm 0.0016}$	0.9082$_{\pm 0.0013}$	0.9062$_{\pm 0.0013}$
	WCAT	0.9134$_{\pm 0.0020}$	0.9144$_{\pm 0.0020}$	0.9141$_{\pm 0.0056}$	0.8291$_{\pm 0.1350}$	0.8298$_{\pm 0.1301}$	0.9217$_{\pm 0.0015}$	0.9212$_{\pm 0.0027}$	0.9194$_{\pm 0.0041}$	0.9167$_{\pm 0.0028}$	0.9155$_{\pm 0.0018}$
	OCM	0.9141$_{\pm 0.0005}$	0.9134$_{\pm 0.0017}$	0.9079$_{\pm 0.0007}$	0.8148$_{\pm 0.1576}$	0.8078$_{\pm 0.1689}$	0.9153$_{\pm 0.0003}$	0.9149$_{\pm 0.0015}$	0.9121$_{\pm 0.0022}$	0.9098$_{\pm 0.0060}$	0.9090$_{\pm 0.0066}$
	ECOC	0.9146$_{\pm 0.0018}$	0.9141$_{\pm 0.0018}$	0.9147$_{\pm 0.0050}$	0.7207$_{\pm 0.3261}$	0.7179$_{\pm 0.3351}$	0.9077$_{\pm 0.0015}$	0.9089$_{\pm 0.0035}$	0.9097$_{\pm 0.0034}$	0.9074$_{\pm 0.0042}$	0.9072$_{\pm 0.0034}$
	ECC	0.9094$_{\pm 0.0003}$	0.9092$_{\pm 0.0007}$	0.9097$_{\pm 0.0005}$	0.8081$_{\pm 0.1751}$	0.8083$_{\pm 0.1744}$	0.9031$_{\pm 0.0003}$	0.9031$_{\pm 0.0003}$	0.9031$_{\pm 0.0003}$	0.9031$_{\pm 0.0003}$	0.9031$_{\pm 0.0003}$
	Ours-U	0.8887$_{\pm 0.0007}$	0.8864$_{\pm 0.0078}$	0.8848$_{\pm 0.0055}$	0.8716$_{\pm 0.0122}$	0.8644$_{\pm 0.0107}$	0.8973$_{\pm 0.0012}$	0.8952$_{\pm 0.0020}$	0.8963$_{\pm 0.0037}$	0.8761$_{\pm 0.0033}$	0.6413$_{\pm 0.3997}$
	Ours-B	**0.9305**$_{\pm 0.0030}$	**0.9264**$_{\pm 0.0022}$	**0.9231**$_{\pm 0.0018}$	**0.9196**$_{\pm 0.0032}$	**0.9151**$_{\pm 0.0019}$	**0.9327**$_{\pm 0.0022}$	**0.9330**$_{\pm 0.0013}$	**0.9304**$_{\pm 0.0049}$	**0.9264**$_{\pm 0.0054}$	**0.9230**$_{\pm 0.0015}$
CIFAR100	None	0.7005$_{\pm 0.0051}$	0.6927$_{\pm 0.0055}$	0.6833$_{\pm 0.0080}$	0.6007$_{\pm 0.1238}$	0.5921$_{\pm 0.1213}$	0.6956$_{\pm 0.0027}$	0.6953$_{\pm 0.0010}$	0.6909$_{\pm 0.0014}$	0.6875$_{\pm 0.0065}$	0.6880$_{\pm 0.0051}$
	WCAT	0.7005$_{\pm 0.0051}$	0.6927$_{\pm 0.0055}$	0.6833$_{\pm 0.0080}$	0.6007$_{\pm 0.1238}$	0.5921$_{\pm 0.1213}$	0.7083$_{\pm 0.0020}$	**0.7056**$_{\pm 0.0030}$	**0.7007**$_{\pm 0.0032}$	**0.6969**$_{\pm 0.0054}$	0.6913$_{\pm 0.0066}$
	OCM	**0.7100**$_{\pm 0.0042}$	**0.7056**$_{\pm 0.0028}$	0.6997$_{\pm 0.0043}$	0.5285$_{\pm 0.2720}$	0.5223$_{\pm 0.2766}$	0.6668$_{\pm 0.0044}$	0.6691$_{\pm 0.0025}$	0.6646$_{\pm 0.0056}$	0.6652$_{\pm 0.0017}$	0.6608$_{\pm 0.0066}$
	ECOC	0.7075$_{\pm 0.0039}$	0.7044$_{\pm 0.0042}$	**0.6999**$_{\pm 0.0040}$	0.5749$_{\pm 0.1883}$	0.5630$_{\pm 0.2013}$	0.6777$_{\pm 0.0021}$	0.6755$_{\pm 0.0036}$	0.6763$_{\pm 0.0026}$	0.6715$_{\pm 0.0039}$	0.6738$_{\pm 0.0013}$
	ECC	0.6680$_{\pm 0.0000}$	0.6678$_{\pm 0.0010}$	0.6682$_{\pm 0.0003}$	0.5441$_{\pm 0.2171}$	0.5441$_{\pm 0.2158}$	0.6897$_{\pm 0.0003}$	0.6895$_{\pm 0.0005}$	0.6890$_{\pm 0.0010}$	0.6886$_{\pm 0.0012}$	0.6873$_{\pm 0.0022}$
	Ours-U	0.6559$_{\pm 0.0073}$	0.6453$_{\pm 0.0086}$	0.6292$_{\pm 0.0086}$	0.5874$_{\pm 0.0222}$	0.5518$_{\pm 0.0339}$	0.6509$_{\pm 0.0043}$	0.6369$_{\pm 0.0205}$	0.6266$_{\pm 0.0227}$	0.4922$_{\pm 0.1623}$	0.4674$_{\pm 0.1714}$
	Ours-B	0.6961$_{\pm 0.0040}$	0.6906$_{\pm 0.0032}$	0.6714$_{\pm 0.0102}$	**0.6418**$_{\pm 0.0158}$	**0.6255**$_{\pm 0.0132}$	**0.7122**$_{\pm 0.0052}$	0.7027$_{\pm 0.0043}$	0.6991$_{\pm 0.0059}$	0.6990$_{\pm 0.0062}$	**0.6946**$_{\pm 0.0081}$
GTSRB	None	0.9817$_{\pm 0.0017}$	0.9795$_{\pm 0.0032}$	0.9775$_{\pm 0.0010}$	0.9191$_{\pm 0.0730}$	0.9076$_{\pm 0.0744}$	0.9823$_{\pm 0.0007}$	0.9821$_{\pm 0.0007}$	0.9702$_{\pm 0.0199}$	0.9701$_{\pm 0.0206}$	0.9696$_{\pm 0.0190}$
	WCAT	0.9873$_{\pm 0.0005}$	0.9862$_{\pm 0.0016}$	0.9841$_{\pm 0.0017}$	0.9491$_{\pm 0.0511}$	0.9476$_{\pm 0.0536}$	0.9824$_{\pm 0.0007}$	0.9822$_{\pm 0.0003}$	0.8816$_{\pm 0.1419}$	0.9115$_{\pm 0.1233}$	0.8748$_{\pm 0.1530}$
	OCM	0.9884$_{\pm 0.0015}$	0.9881$_{\pm 0.0010}$	0.9871$_{\pm 0.0003}$	0.9632$_{\pm 0.0338}$	0.9577$_{\pm 0.0446}$	0.9851$_{\pm 0.0003}$	0.9849$_{\pm 0.0005}$	0.9854$_{\pm 0.0007}$	0.9846$_{\pm 0.0003}$	0.9847$_{\pm 0.0007}$
	ECOC	0.9847$_{\pm 0.0007}$	0.9842$_{\pm 0.0012}$	0.9836$_{\pm 0.0033}$	0.9595$_{\pm 0.0334}$	0.9517$_{\pm 0.0444}$	0.9868$_{\pm 0.0005}$	0.9866$_{\pm 0.0003}$	0.9697$_{\pm 0.0242}$	0.9749$_{\pm 0.0206}$	0.9757$_{\pm 0.0205}$
	ECC	0.9849$_{\pm 0.0000}$	0.9849$_{\pm 0.0000}$	0.9849$_{\pm 0.0000}$	0.9670$_{\pm 0.0252}$	0.9668$_{\pm 0.0255}$	0.9761$_{\pm 0.0000}$	0.9761$_{\pm 0.0000}$	0.6995$_{\pm 0.3912}$	0.7909$_{\pm 0.3204}$	0.7910$_{\pm 0.3201}$
	Ours-U	0.9692$_{\pm 0.0073}$	0.9679$_{\pm 0.0068}$	0.9639$_{\pm 0.0098}$	0.9538$_{\pm 0.0117}$	0.9388$_{\pm 0.0178}$	0.9743$_{\pm 0.0033}$	0.9634$_{\pm 0.0174}$	0.9606$_{\pm 0.0170}$	0.8496$_{\pm 0.1646}$	0.8089$_{\pm 0.1892}$
	Ours-B	**0.9901**$_{\pm 0.0005}$	**0.9903**$_{\pm 0.0008}$	**0.9894**$_{\pm 0.0014}$	**0.9805**$_{\pm 0.0014}$	**0.9747**$_{\pm 0.0014}$	**0.9993**$_{\pm 0.0004}$	**0.9996**$_{\pm 0.0002}$	**0.9994**$_{\pm 0.0001}$	**0.9996**$_{\pm 0.0001}$	**0.9992**$_{\pm 0.0004}$
ImageNet1K	None	0.6743$_{\pm 0.0169}$	0.6345$_{\pm 0.0059}$	0.5160$_{\pm 0.0500}$	0.1632$_{\pm 0.1405}$	0.0931$_{\pm 0.0852}$	**0.6387**$_{\pm 0.0043}$	0.6082$_{\pm 0.0097}$	0.5003$_{\pm 0.0887}$	0.3351$_{\pm 0.0736}$	0.2633$_{\pm 0.0880}$
	WCAT	0.6673$_{\pm 0.0175}$	0.6390$_{\pm 0.0132}$	0.5254$_{\pm 0.0528}$	0.1908$_{\pm 0.1690}$	0.1245$_{\pm 0.1108}$	0.6283$_{\pm 0.0069}$	0.5991$_{\pm 0.0107}$	0.4917$_{\pm 0.0876}$	0.3213$_{\pm 0.0424}$	0.2399$_{\pm 0.0714}$
	OCM	0.5187$_{\pm 0.0386}$	0.5024$_{\pm 0.0370}$	0.4357$_{\pm 0.0738}$	0.1468$_{\pm 0.1292}$	0.1105$_{\pm 0.1007}$	0.5599$_{\pm 0.0043}$	0.5583$_{\pm 0.0041}$	0.5340$_{\pm 0.0150}$	0.4946$_{\pm 0.0097}$	0.4650$_{\pm 0.0361}$
	ECOC	0.4535$_{\pm 0.0060}$	0.4419$_{\pm 0.0056}$	0.3965$_{\pm 0.0265}$	0.1934$_{\pm 0.1674}$	0.1489$_{\pm 0.1318}$	0.5737$_{\pm 0.0014}$	0.5690$_{\pm 0.0064}$	0.5452$_{\pm 0.0220}$	0.4956$_{\pm 0.0157}$	0.4551$_{\pm 0.0454}$
	ECC	0.6138$_{\pm 0.0005}$	0.6135$_{\pm 0.0003}$	0.6110$_{\pm 0.0015}$	0.4093$_{\pm 0.3524}$	0.4103$_{\pm 0.3532}$	0.5811$_{\pm 0.0013}$	0.5812$_{\pm 0.0015}$	**0.5811**$_{\pm 0.0005}$	0.4907$_{\pm 0.0010}$	0.4508$_{\pm 0.0003}$
	Ours-U	0.6426$_{\pm 0.0055}$	0.6159$_{\pm 0.0265}$	0.5913$_{\pm 0.0319}$	0.5254$_{\pm 0.0273}$	0.4904$_{\pm 0.0340}$	0.6107$_{\pm 0.0003}$	0.5912$_{\pm 0.0003}$	0.5610$_{\pm 0.0005}$	0.0011$_{\pm 0.0007}$	0.0012$_{\pm 0.0003}$
	Ours-B	**0.6929**$_{\pm 0.0035}$	**0.6780**$_{\pm 0.0159}$	**0.6332**$_{\pm 0.0121}$	**0.5454**$_{\pm 0.0330}$	**0.5098**$_{\pm 0.0236}$	0.6214$_{\pm 0.0040}$	**0.6088**$_{\pm 0.0071}$	0.5679$_{\pm 0.0107}$	**0.5053**$_{\pm 0.0610}$	**0.4651**$_{\pm 0.0381}$
SVHN	None	0.9546$_{\pm 0.0007}$	0.9543$_{\pm 0.0016}$	0.9556$_{\pm 0.0005}$	0.9541$_{\pm 0.0027}$	0.9548$_{\pm 0.0024}$	0.9565$_{\pm 0.0010}$	0.9573$_{\pm 0.0003}$	0.9558$_{\pm 0.0017}$	0.9561$_{\pm 0.0022}$	0.9565$_{\pm 0.0015}$
	WCAT	0.9539$_{\pm 0.0010}$	0.9543$_{\pm 0.0012}$	0.9534$_{\pm 0.0010}$	0.9531$_{\pm 0.0020}$	0.9528$_{\pm 0.0007}$	0.9565$_{\pm 0.0015}$	0.9565$_{\pm 0.0007}$	0.9562$_{\pm 0.0017}$	0.9559$_{\pm 0.0037}$	0.9556$_{\pm 0.0030}$
	OCM	0.9619$_{\pm 0.0005}$	0.9614$_{\pm 0.0010}$	0.9622$_{\pm 0.0003}$	0.9591$_{\pm 0.0053}$	0.9583$_{\pm 0.0059}$	0.9648$_{\pm 0.0007}$	0.9653$_{\pm 0.0007}$	0.9639$_{\pm 0.0007}$	0.9622$_{\pm 0.0020}$	0.9619$_{\pm 0.0021}$
	ECOC	0.9646$_{\pm 0.0003}$	0.9645$_{\pm 0.0015}$	0.9639$_{\pm 0.0015}$	0.9597$_{\pm 0.0052}$	0.9604$_{\pm 0.0045}$	0.9658$_{\pm 0.0010}$	0.9661$_{\pm 0.0010}$	0.9652$_{\pm 0.0010}$	0.9661$_{\pm 0.0010}$	0.9644$_{\pm 0.0018}$
	ECC	0.9568$_{\pm 0.0003}$	0.9568$_{\pm 0.0003}$	0.9568$_{\pm 0.0003}$	0.9548$_{\pm 0.0024}$	0.9552$_{\pm 0.0018}$	0.9565$_{\pm 0.0000}$	0.9565$_{\pm 0.0000}$	0.9565$_{\pm 0.0000}$	0.9565$_{\pm 0.0000}$	0.9565$_{\pm 0.0000}$
	Ours-U	0.9531$_{\pm 0.0010}$	0.9523$_{\pm 0.0020}$	0.9543$_{\pm 0.0020}$	0.9538$_{\pm 0.0023}$	0.9499$_{\pm 0.0020}$	0.9562$_{\pm 0.0057}$	0.9577$_{\pm 0.0057}$	0.9556$_{\pm 0.0047}$	0.8577$_{\pm 0.1145}$	0.8035$_{\pm 0.2002}$
	Ours-B	**0.9805**$_{\pm 0.0017}$	**0.9782**$_{\pm 0.0020}$	**0.9788**$_{\pm 0.0019}$	**0.9762**$_{\pm 0.0015}$	**0.9749**$_{\pm 0.0011}$	**0.9839**$_{\pm 0.0004}$	**0.9826**$_{\pm 0.0022}$	**0.9824**$_{\pm 0.0004}$	**0.9812**$_{\pm 0.0013}$	**0.9809**$_{\pm 0.0011}$

Threat evaluation protocol. For *random bit-flip robustness*, we evaluate models under cumulative random faults at a fixed BER. At each step, a new set of random bit flips is injected into the current corrupted model, recovery is applied for recovery-based methods, and robust accuracy is measured. The recovered model state is then carried forward to the next step, so faults accumulate over time rather than being reinitialized at each evaluation point. For *standard adversarial robustness*, we consider a fixed bit-flip budget and optimize attack objectives tailored to each defense. For standard classifiers and most defense methods, the attacker optimizes the classification objective in output-logit space. For OCM- and ECOC-based methods, attacks are optimized in their corresponding code-aligned class-score space so that the objective remains consistent with the model's decoding rule. For stealthy attacks, the objective additionally promotes targeted misclassification on triggered inputs while preserving clean-data performance. After all selected bit flips are applied, recovery is performed for recovery-based methods, and robust accuracy is measured on the resulting attacked model. For *adaptive adversarial robustness*, we evaluate stronger attacks that explicitly account for the recovery mechanism of the defense by combining the objective of each existing attack with the adaptive objective introduced in Section 4. In this setting, the attacker is aware of the recovery procedure used by both ECC-based methods and our method, and selects bit flips that remain effective after recovery is applied. For destructive attacks, we use the destructive adaptive objective, while for stealthy attacks, we use the corresponding stealthy adaptive objective. This setting directly tests whether the recovery mechanism itself can be bypassed or neutralized by an informed adversary.

5.2 Experimental Results

Does the proposed method provide model- and precision-agnostic robustness to cumulative random bit-flip threats? Table 1–3 show that the proposed method consistently provides the strongest robustness to cumulative random faults across datasets, architectures, and numerical precisions. In FLOAT32, existing fault-tolerance methods deteriorate rapidly as errors accumulate and often collapse to near-random accuracy, whereas both variants of our method remain highly stable over many corruption steps. This advantage is particularly evident in Table 1, where **Ours-B** achieves the highest robust accuracy in nearly all settings. Table 2 further shows that the same behavior extends to ViT-B/16, indicating that the proposed recovery principle is not limited to CNNs and generalizes to large Transformer-based architectures as well. In INT8 (Table 3), where quantization itself already offers a degree of numerical robustness, our method still attains the best or near-best performance in

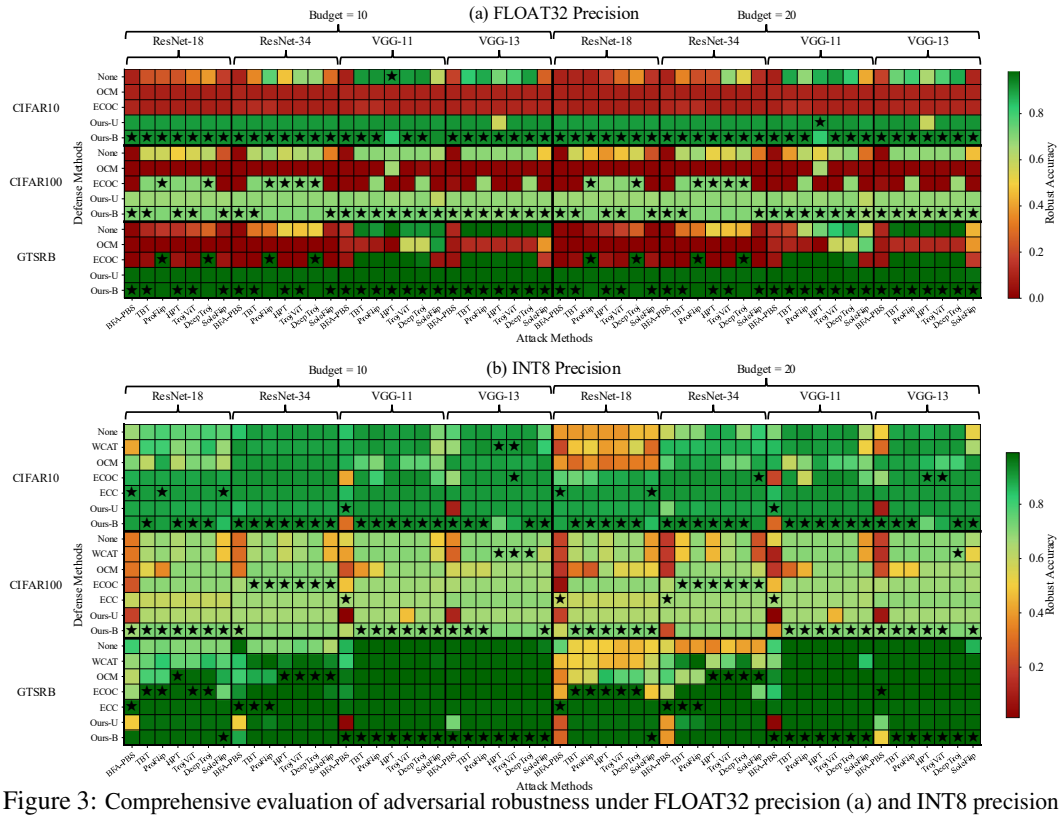

Figure 3: Comprehensive evaluation of adversarial robustness under FLOAT32 precision (a) and INT8 precision (b) for bit-flip budgets of 10 and 20. For stealthy attacks, robust accuracy is measured on triggered inputs. "⋆" denotes the best robust accuracy in each setting.

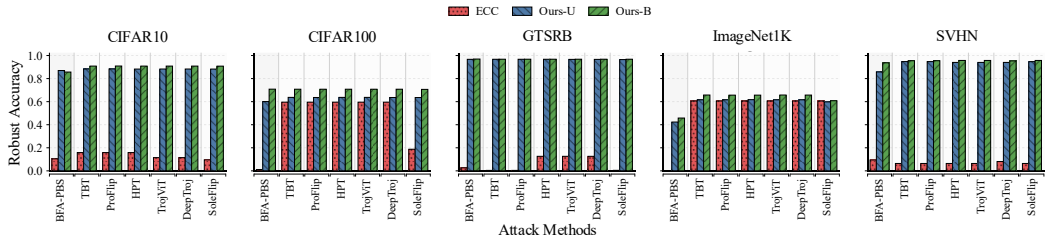

Figure 4: Recovery-aware adaptive adversarial attack evaluation for recovery-based methods on INT8 ResNet-18 at a bit-flip budget of 5. For stealthy attacks, robust accuracy is measured on triggered inputs.

all ten architecture–dataset settings, including the additional comparisons with RQUANT+Clipping, RandBET, and AdvBET-FO in Table 8. Notably, ECC exhibits substantially larger variance in several settings, suggesting that its recovery mechanism can itself become fragile under accumulated random corruption due to concentrated failure points of ECC. By contrast, our method remains both accurate and stable. These results indicate that the proposed approach is both model-agnostic and precision-agnostic, while providing strong robustness to cumulative random bit-flip faults and demonstrating broad generalizability across architectures and numerical settings.

Does the proposed method provide model- and precision-agnostic robustness under adversarial bit-flip threats? Figure 3 shows that our method consistently delivers the strongest adversarial robustness across datasets, architectures, precisions, and attack families. The margin is especially pronounced in FLOAT32, where competing methods, including more sophisticated code-based defenses such as ECOC, often degrade to near-zero robust accuracy even under moderate attack budgets, while our method maintains strong performance across both destructive and stealthy attacks. In INT8, the performance gap is naturally narrower because quantized models already benefit from

greater numerical stability; nevertheless, our method still achieves the highest robust accuracy in most settings. Existing fault-tolerance methods such as WCAT and OCM degrade substantially as the bit-flip budget increases, whereas our recovery-based design remains consistently strong. These results indicate that the proposed method is effective not only against random corruption but also against highly optimized adversarial bit-flip attacks, and that this advantage persists across both precisions and model families.

Does the proposed method remain robust under stronger adaptive attack threats? Figure 4 compares recovery-based defenses under adaptive attacks that explicitly account for the recovery mechanism itself. Our method substantially outperforms ECC across datasets and attacks, showing that it remains effective even when the adversary is aware of the defense and optimizes against its post-recovery behavior. By contrast, ECC is much more readily neutralized under adaptive optimization. This result directly supports our central claim that a robust recovery mechanism should avoid concentrated vulnerable structures that can be disabled by only a small number of carefully chosen flips. In contrast to ECC, our method promotes recoverability by reinforcing structural properties that naturally arise in trained weight distributions, leading to a more distributed and less concentrated vulnerability surface, as suggested by these empirical results. In the additional threshold-evading evaluation on CIFAR-10/ResNet-18/INT8, Ours-B retains 0.847 accuracy after 100 flips, whereas the undefended baseline drops to 0.107 after only five unrestricted flips (Appendix D). Given the time and effort required to induce even a single targeted bit flip in practice, this demonstrates substantial practical robustness against bit-flip injection attacks.

Is balanced zero-sum projection preferable to the unbalanced variant? Across nearly all experiments, **Ours-B** consistently performs better than, or at least matches, **Ours-U**. This trend suggests that the balanced formulation is preferable in practice. Empirically, assigning special structural roles to the final row and column, as in the unbalanced formulation, appears to introduce an optimization asymmetry that slightly weakens robustness. The balanced formulation avoids this asymmetry and consequently delivers the strongest overall performance.

6 Conclusion

We introduced a simple training-based recovery framework that endows DNN weights with recoverable structure through differentiable block-wise zero-sum projection. By reinforcing the approximately zero-centered structure of naturally trained weights, our method enables efficient corruption detection and recovery without additional weight-storage overhead. The framework generalizes across architectures, tasks, and numerical precisions, remains compatible with standard tensor computation, and largely preserves clean performance. Evaluations under cumulative random faults, adversarial bit flips, and recovery-aware adaptive attacks demonstrate consistent robustness gains, highlighting the potential of learning recoverability directly within model weights.

Limitations and future work. The current method constructs $K \times K$ blocks by flattening weight tensors. Developing more effective grouping schemes that preserve computational efficiency, together with principled block-size selection across architectures, precisions, and fault rates, remains a direction for future work. Our adaptive evaluation is also limited: neither greedy recovery-aware rescoring nor per-flip threshold evasion exhaustively covers jointly optimized residual-canceling patterns. Although simultaneous multi-bit fault injection is challenging in practice, attackers with sufficient spatial and temporal control could induce such patterns. Characterizing and strengthening the robustness of zero-sum recovery against these coordinated attacks remains an open problem.

Acknowledgments and Disclosure of Funding

This work was supported in part by the DARPA Young Faculty Award, the National Science Foundation (NSF) under Grants #2127780, #2319198, #2321840, #2312517, and #2235472, the Semiconductor Research Corporation (SRC), the Office of Naval Research through the Young Investigator Program Award, and Grants #N00014-21-1-2225 and N00014-24-1-2547, Army Research Office Grant #W911NF2410360. Additionally, support was provided by the Air Force Office of Scientific Research under Award #FA9550-22-1-0253.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to flip a bit? In *2010 IEEE 16th International On-Line Testing Symposium*, pages 235–239. IEEE, 2010.

[3] Sabbir Ahmed, Ranyang Zhou, Shaahin Angizi, and Adnan Siraj Rakin. Deep-troj: An inference stage trojan insertion algorithm through efficient weight replacement attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24810–24819, 2024.

[4] Jiawang Bai, Kuofeng Gao, Dihong Gong, Shu-Tao Xia, Zhifeng Li, and Wei Liu. Hardly perceptible trojan attack against neural networks with bit flips. In *European Conference on Computer Vision*, pages 104–121. Springer, 2022.

[5] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012.

[6] Sanguhn Cha, O Seongil, Hyunsung Shin, Sangjoon Hwang, Kwangil Park, Seong Jin Jang, Joo Sun Choi, Gyo Young Jin, Young Hoon Son, Hyunyoon Cho, et al. Defect analysis and cost-effective resilience architecture for future dram devices. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 61–72. IEEE, 2017.

[7] Nandhini Chandramoorthy, Karthik Swaminathan, Martin Cochet, Arun Paidimarri, Schuyler Eldridge, Rajiv V Joshi, Matthew M Ziegler, Alper Buyuktosunoglu, and Pradip Bose. Resilient low voltage accelerators for high energy efficiency. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 147–158. IEEE, 2019.

[8] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Proflip: Targeted trojan attack with progressive bit flips. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7718–7727, 2021.

[9] Yanping Chen, Yuan Hao, Thanawin Rakthanmanon, Jesin Zakaria, Bing Hu, and Eamonn Keogh. A general framework for never-ending learning from time series streams. *Data mining and knowledge discovery*, 29(6):1622–1664, 2015.

[10] Kamran Chitsaz, Goncalo Mordido, Jean-Pierre David, and François Leduc-Primeau. Training dnns resilient to adversarial and random bit-flips by learning quantization ranges. *Transactions on Machine Learning Research*, 2023.

[11] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Advances in neural information processing systems*, 36:49250–49267, 2023.

[12] Jianshuo Dong, Han Qiu, Yiming Li, Tianwei Zhang, Yuanjie Li, Zeqi Lai, Chao Zhang, and Shu-Tao Xia. One-bit flip is all you need: When bit-flip attack meets model training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4688–4698, 2023.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[14] Shrikanth Ganapathy, John Kalamatianos, Keith Kasprak, and Steven Raasch. On characterizing near-threshold sram failures in finfet technology. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.

[15] Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012.

[16] Hui Guan, Lin Ning, Zhen Lin, Xipeng Shen, Huiyang Zhou, and Seung-Hwan Lim. In-place zero-space memory protection for cnn. *Advances in Neural Information Processing Systems*, 32, 2019.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[18] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraș. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 497–514, 2019.

[19] Mu-Yue Hsiao. A class of optimal minimum odd-weight-column sec-ded codes. *IBM Journal of Research and Development*, 14(4):395–401, 1970.

[20] Younis Ibrahim, Haibin Wang, Man Bai, Zhi Liu, Jianan Wang, Zhiming Yang, and Zhengming Chen. Soft error resilience of deep residual networks for object recognition. *IEEE Access*, 8:19490–19503, 2020.

[21] Muhammad Imran, Taehyun Kwon, Jung Min You, and Joon-Sung Yang. Flipcy: Efficient pattern redistribution for enhancing mlc pcm reliability and storage density. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7. IEEE, 2019.

[22] Patrick Jattke, Victor Van Der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. Blacksmith: Scalable rowhammering in the frequency domain. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 716–734. IEEE, 2022.

[23] Jae-San Kim and Joon-Sung Yang. Dris-3: Deep neural network reliability improvement scheme in 3d die-stacked memory based on fault analysis. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[24] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[26] Shamik Kundu, Kanad Basu, Mehdi Sadi, Twisha Titirsha, Shihao Song, Anup Das, and Ujjwal Guin. Special session: Reliability analysis for ai/ml hardware. In *2021 IEEE 39th VLSI Test Symposium (VTS)*, pages 1–10. IEEE, 2021.

[27] Kevin Lange, Federico Fontana, Francesco Rossi, Mattia Varile, and Giovanni Apruzzese. Machine learning in space: Surveying the robustness of on-board ml models to radiation. In *2024 IEEE Space Computing Conference (SCC)*, pages 51–64. IEEE, 2024.

[28] Seo-Seok Lee and Joon-Sung Yang. Value-aware parity insertion ecc for fault-tolerant deep neural network. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 724–729. IEEE, 2022.

[29] Shaofeng Li, Xinyu Wang, Minhui Xue, Haojin Zhu, Zhi Zhang, Yansong Gao, Wen Wu, and Xuemin Sherman Shen. Yes,{One-Bit-Flip} matters! universal {DNN} model inference depletion with runtime code fault injection. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1315–1330, 2024.

[30] Xiang Li, Lannan Luo, and Qiang Zeng. Backdoor attacks on neural networks via one-bit flip. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4328–4338, 2025.

[31] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256kb memory. *Advances in Neural Information Processing Systems*, 35:22941–22954, 2022.

[32] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 131–138. IEEE, 2017.

[33] Szymon Migacz. 8-bit inference with tensorrt. In *GPU technology conference*, volume 2, page 5, 2017.

[34] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 4. Granada, 2011.

[35] Ozan Özdenizci and Robert Legenstein. Improving robustness against stealthy weight bit-flip attacks by output code matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13388–13397, 2022.

[36] Minghai Qin, Chao Sun, and Dejan Vucinic. Robustness of neural networks against storage media errors. *arXiv preprint arXiv:1709.06173*, 2017.

[37] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

[38] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *2022 IEEE symposium on security and privacy (SP)*, pages 1157–1174. IEEE, 2022.

[39] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1211–1220, 2019.

[40] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Tbt: Targeted neural network attack with bit trojan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13198–13207, 2020.

[41] Adnan Siraj Rakin, Yukui Luo, Xiaolin Xu, and Deliang Fan. {Deep-Dup}: An adversarial weight duplication attack framework to crush deep neural network in {Multi-Tenant}{FPGA}. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1919–1936, 2021.

[42] Rodrigo P Rocha, Marco Zorzi, and Maurizio Corbetta. Role of homeostatic plasticity in critical brain dynamics following focal stroke lesions. *Scientific Reports*, 14(1):31631, 2024.

[43] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[44] Eyal Rozenfeld, Nadine Ehmann, Julia E Manoim, Robert J Kittel, and Moshe Parnas. Homeostatic synaptic plasticity rescues neural coding reliability. *Nature Communications*, 14(1):2993, 2023.

[45] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[46] Bodo Selmke, Stefan Brummer, Johann Heyszl, and Georg Sigl. Precise laser fault injections into 90 nm and 45 nm sram-cells. In *International Conference on Smart Card Research and Advanced Applications*, pages 193–205. Springer, 2015.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[48] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks*, pages 1453–1460. IEEE, 2011.

[49] David Stutz, Nandhini Chandramoorthy, Matthias Hein, and Bernt Schiele. Random and adversarial bit error robustness: Energy-efficient and secure dnn accelerators. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3632–3647, 2022.

[50] Hao-Lun Sun, Lei Hsiung, Nandhini Chandramoorthy, Pin-Yu Chen, and Tsung-Yi Ho. Neuralfuse: Learning to recover the accuracy of access-limited neural network inference in low-voltage regimes. *Advances in Neural Information Processing Systems*, 37:54558–54594, 2024.

[51] Qiaoyu Tang, Jiawei Chen, Zhuoqun Li, Bowen Yu, Yaojie Lu, Cheng Fu, Haiyang Yu, Hongyu Lin, Fei Huang, Ben He, et al. Self-retrieval: End-to-end information retrieval with one large language model. *Advances in Neural Information Processing Systems*, 37:63510–63533, 2024.

[52] M Caner Tol, Saad Islam, Andrew J Adiletta, Berk Sunar, and Ziming Zhang. Don't knock! rowhammer at the backdoor of dnn models. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 109–122. IEEE, 2023.

[53] Gina G Turrigiano. The self-tuning neuron: synaptic scaling of excitatory synapses. *Cell*, 135(3):422–435, 2008.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[55] Jialai Wang, Yuxiao Wu, Weiye Xu, Yating Huang, Chao Zhang, Zongpeng Li, Mingwei Xu, and Zhenkai Liang. Your scale factors are my weapon: Targeted bit-flip attacks on vision transformers via scale factor manipulation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 20103–20112, 2025.

[56] Jialai Wang, Ziyuan Zhang, Meiqi Wang, Han Qiu, Tianwei Zhang, Qi Li, Zongpeng Li, Tao Wei, and Chao Zhang. Aegis: Mitigating targeted bit-flip attacks against deep neural networks. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2329–2346, 2023.

[57] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. {DeepHammer}: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1463–1480, 2020.

[58] Anlan Yu, Shusen Jing, Ning Lyu, Wujie Wen, and Zhiyuan Yan. Error correction output codes for robust neural networks against weight-errors: A neural tangent kernel point of view. *Advances in Neural Information Processing Systems*, 37:82493–82513, 2024.

[59] Sanggeon Yun, Ryozo Masukawa, SungHeon Jeong, Mahdi Imani, and Mohsen Imani. Neurohash: A hyperdimensional neuro-symbolic framework for spatially-aware image hashing and retrieval. In *ICASSP 2026-2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8597–8601. IEEE, 2026.

[60] Sanggeon Yun, Hyunwoo Oh, Ryozo Masukawa, and Mohsen Imani. Decohd: Decomposed hyperdimensional classification under extreme memory budgets. In *2026 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7. IEEE, 2026.

[61] Sanggeon Yun, Hyunwoo Oh, Ryozo Masukawa, Pietro Mercati, Nathaniel D Bastian, and Mohsen Imani. Loghd: Robust compression of hyperdimensional classifiers via logarithmic class-axis reduction. In *2026 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7. IEEE, 2026.

[62] Mengxin Zheng, Qian Lou, and Lei Jiang. Trojvit: Trojan insertion in vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4025–4034, 2023.

Appendix

A Background

A.1 Bit-flip Threats

Random Bit Errors in DNNs. As DNNs are increasingly deployed in edge devices, accelerators, and other resource-constrained platforms, model parameters stored in SRAM or DRAM are exposed to non-ideal operating conditions such as aggressive low-voltage execution, power and thermal fluctuations, device aging, and environmental disturbances, including radiation-induced faults in safety-critical or high-altitude and space settings, all of which can induce spontaneous memory errors [7, 27, 50]. This issue is particularly important because modern deployment settings often prioritize energy efficiency and compact hardware footprints, which further shrink reliability margins and make memory corruption more likely in practice [7, 27]. As a result, random bit errors have become an increasingly important obstacle to the safe and reliable use of DNNs in real applications. Existing fault-tolerance approaches can mitigate the immediate impact of a limited number of corruptions, but they fundamentally do not reverse the accumulation of errors in stored weights; as corruptions continue to accumulate, the deployed model progressively drifts away from its clean parameter state and eventually suffers severe performance degradation, motivating the need for recovery-oriented mechanisms in addition to tolerance alone [50, 27].

Adversarial Bit-flip Attacks. Adversarial bit flips arise when an attacker intentionally induces a small number of weight-bit corruptions in a deployed DNN by exploiting physical memory-fault mechanisms. Prior work has shown that disturbance-based techniques such as RowHammer can corrupt nearby DRAM rows without direct access to the victim cells [24], while laser fault injection can localize faults to individual or near-individual SRAM cells under carefully calibrated conditions [5, 46, 2]. Although such fault-injection primitives are powerful, they are also costly, constrained, and hardware dependent: reliably inducing faults at chosen locations can take substantial time, and realizing multiple prescribed flips is often considerably harder than inducing a single corruption [22, 56, 12, 29, 41, 38, 57]. These practical constraints have motivated a line of *bit-efficient* attacks tailored to DNNs. Early work showed that carefully chosen flips can catastrophically collapse model accuracy with only a handful of corruptions [39], and subsequent studies extended this idea to more targeted and stealthy settings that preserve benign performance while inducing attacker-controlled behavior only under specific triggers [40, 8, 4, 52, 62, 3, 30]. As a result, adversarial bit flips have become an important part of the reliability and security landscape for deployed DNNs, since modern attack algorithms can cause severe or highly targeted behavioral changes under extremely small bit-flip budgets.

A.2 Defense Methods

Existing defenses against weight corruption can be broadly divided into *fault tolerance* and *fault recovery*. Fault-tolerance methods aim to preserve predictive performance despite corrupted parameters, whereas fault-recovery methods seek to detect and correct the corrupted weights themselves after faults occur.

Bit-flip-aware Training Methods. A straightforward fault-tolerance strategy is to incorporate the fault model directly into training. Earlier work by Stutz et al. [49] includes robust fixed-point quantization with clipping (RQUANT+Clipping), random bit error training (RandBET), and adversarial bit error training, whose first-order variant we denote by AdvBET-FO. A more recent approach, *weight clipping-aware training* (WCAT), learns tighter quantization ranges, optionally incorporating bit-flip noise during training, to reduce the deployed quantized model's sensitivity to both random and adversarial bit flips [10]. We use WCAT as our primary training-based baseline, as it is a more recent method in this broader line of work; additional comparisons with all four methods are reported in Table 8 in the appendix. *NeuralFuse* takes a different route by training an auxiliary input generator that transforms images into representations that remain classifiable even when the backbone is corrupted by random SRAM faults [50]. These approaches can improve robustness to limited one-shot corruptions, but they remain fundamentally tolerance-oriented: they do not restore corrupted parameters and therefore degrade as faults accumulate over time. They are also weak against white-box adversarial bit-flip attacks, since an attacker can still target the most vulnerable

bits of the deployed model directly. In addition, their robustness gains are often constrained in scope: WCAT reports that explicit bit-flip-aware training contributes only modestly beyond clipping itself [10], while NeuralFuse relies on a relatively heavy auxiliary generator and does not account for faults in that generator. More broadly, such methods are naturally tailored to quantized settings and do not transfer cleanly to floating-point representations, where even a single exponent-bit corruption can induce disproportionately large parameter changes.

OCM-based Methods. A stronger fault-tolerance direction modifies the output representation rather than the internal weights. *Output code matching* (OCM) replaces the standard one-hot target with structured codewords, such as Hadamard codes, and performs prediction by matching outputs to codewords instead of taking the largest logit [35]. Recent ECOC variants generalize this idea by using non-one-hot output codes with larger error-correcting capability, yielding improved robustness to weight corruption relative to conventional classifiers [58]. These methods are attractive, as they are simple, architecture-compatible, and often effective against both random and adversarial corruptions. However, they also remain tolerance-oriented: they do not recover the corrupted weights, are vulnerable to cumulative faults, and offer limited protection under white-box attack settings where the adversary optimizes directly over the deployed parameters. They also introduce output-layer overhead by expanding the classifier dimension beyond the number of classes, and much of their protection is concentrated in the final layer [55], which limits the extent of robustness that can be achieved in the full network.

ECC-based Methods. ECC-based approaches constitute the canonical fault-recovery family. Their goal is to embed error-detecting and error-correcting structures into the stored model parameters so that corrupted bits can be identified and corrected at deployment time [36, 16, 28]. A representative example is *in-place ECC* [16], which trains the model so that selected bits within each weight become functionally insignificant, and then replaces those bits with ECC parity information, thereby enabling recovery without additional storage overhead. Compared with tolerance-only methods, ECC-based approaches offer explicit recovery capability and are inherently better suited to cumulative corruption. However, this protection comes at a cost: the construction is highly specialized, making it difficult to generalize across precisions and architectures. Moreover, the encoded structure itself can become an attack surface, since an adversary may invalidate recoverability by corrupting at least two chosen bits of the code. Finally, ECC decoding is imposed externally on top of DNN computation rather than arising naturally from the model's tensor operations, which complicates efficient deployment.

A.3 Model Precisions and Bit-flip

Floating-Point and Quantized Precision. The effect of a bit flip depends fundamentally on how model parameters are represented in memory. In floating-point deployment, each weight is stored using a sign bit, exponent bits, and mantissa bits. For example, under IEEE-style formats, a weight can be written as

$$w = (-1)^s \cdot 2^{e-b} \cdot (1.m), \tag{18}$$

where s is the sign bit, e is the stored exponent, b is the exponent bias, and m denotes the mantissa. A single bit flip in the sign or exponent can therefore induce a disproportionately large change in the represented value. By contrast, in post-quantized deployment, weights are first mapped to low-bit integers and then reconstructed through a scale factor. In standard implementation [33], a full-precision weight tensor W is quantized as

$$Q = \mathrm{clip}\left(\mathrm{round}\left(\frac{W}{\Delta}\right), -q_{\max}, q_{\max}\right), \qquad \hat{W} = \Delta Q, \tag{19}$$

where Q denotes the quantized integer tensor, $\hat{W}$ the dequantized tensor used at deployment, $q_{\max} = 2^{b-1} - 1$ the maximum magnitude representable under signed b-bit quantization, and Δ the per-layer quantization scale, typically chosen from the maximum absolute value of W in that layer. Under this representation, individual weight perturbations are bounded by the layer scale, making quantized models naturally less sensitive to many single-bit perturbations than floating-point models [18, 39, 40, 30].

Precisions in Attacks and Defenses. Precision strongly shapes both vulnerability and defense applicability. Floating-point models are typically more fragile under bit corruption, since flipping

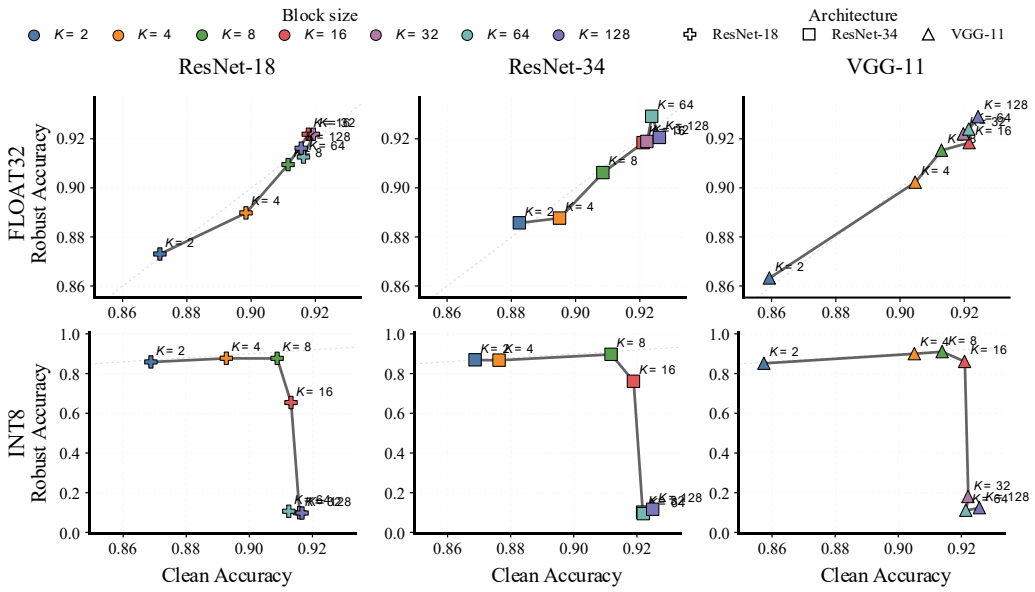

Figure 5: Effect of block size K on clean accuracy and cumulative random-fault robustness on CIFAR10 for ResNet-18, ResNet-34, and VGG-11. Each point corresponds to a different block size. In FLOAT32, larger K generally improves both clean and robust accuracy before saturating. In INT8, both metrics generally improve up to about $K = 8$, beyond which gains in clean accuracy saturate while robust accuracy drops substantially around $K = 32$, suggesting that $K \approx 8$ offers the best balance between clean and robust accuracy for quantized models.

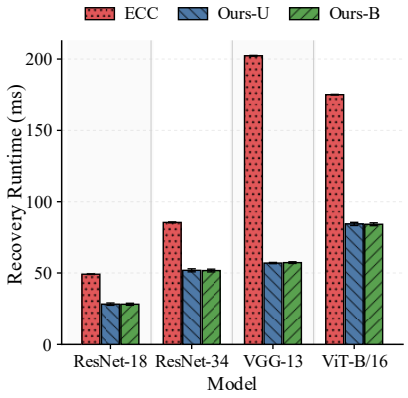

Figure 6: Recovery runtime overhead in INT8 across different architectures. Bars show mean recovery latency in milliseconds, with error bars indicating standard deviation over 30 repeated measurements. Both variants of our method are consistently faster than ECC, with the gap becoming larger on larger models such as VGG-13 and ViT-B/16.

a sign or exponent bit can cause a large weight perturbation, whereas quantized models bound per-weight error by the quantization scale, which is one reason prior defenses often focus on INT8-like settings [18, 39, 40, 30]. Yet this robustness introduces another attack surface: the layer scale. Since one scale governs many weights, corrupting it can simultaneously distort an entire layer, enabling highly bit-efficient attacks [55]. This precision dependence also limits existing defenses, many of which are tailored to specific quantized formats, do not extend naturally to floating-point deployment, and often overlook shared scale parameters. A practical defense should therefore operate across both floating-point and quantized models while accounting for all deployed parameters that affect inference, including quantization scales.

B Effect of Block Size

We further study how the block size K affects the relationship between clean accuracy and robustness. Figure 5 reports this analysis on CIFAR10 for ResNet-18, ResNet-34, and VGG-11 under cumulative random bit-flip faults. Interestingly, the relationship is not a simple robustness–accuracy trade-off. In FLOAT32, clean accuracy and robust accuracy are positively correlated across the tested range: increasing K generally improves both metrics, producing an almost linear trend for smaller to moderate block sizes. This improvement begins to saturate around $K = 16$ to $K = 32$, suggesting that larger block sizes do not necessarily reduce robustness in the floating-point setting, but instead offer diminishing returns beyond a certain point. In INT8, we observe a related but qualitatively different pattern. Clean accuracy and robust accuracy again improve together as K increases from small values, but the trend begins to bend around $K \approx 8$. Beyond this point, gains in clean accuracy saturate, while robust accuracy drops substantially around $K = 32$. This pattern suggests a more clearly defined optimal block-size region for quantized models. Across the tested architectures and range of values, this optimum consistently appears near $K = 8$, which supports our default choice in the main experiments.

C Recovery Runtime Overhead

We also evaluate the computational overhead of recovery in the INT8 setting. For ECC, recovery is implemented using a standard accelerated SECDED-style syndrome-decoding procedure over protected quantized blocks, following the conventional parity-check/syndrome-correction formulation used in efficient SECDED decoders [19]. For our method, by contrast, recovery is implemented entirely with batched tensor operations and executed on the GPU, so the reported runtime reflects a fully parallelized realization of the proposed recovery rule. As shown in Figure 6, both variants of our method consistently require substantially less recovery time than ECC across all evaluated architectures. On smaller CNNs such as ResNet-18 and ResNet-34, our method already provides a clear reduction in latency, and this advantage becomes even more pronounced for larger models such as VGG-13 and ViT-B/16. Notably, **Ours-U** and **Ours-B** exhibit nearly identical runtime, since both share the same recovery mechanism and differ only in the projection used during training. Overall, these results show that the proposed method offers a substantially better robustness–efficiency trade-off than ECC, while remaining effective across both CNN and Transformer architectures.

Deployed full-model scanning and recovery cost. Beyond the recovery overhead measured above, we assess deployment costs by examining full-model scanning and correction alongside ordinary inference latency. Since fault locations are unknown, recovery needs to scan all protected weights and write corrected values back to the deployed representation. On ResNet-18/INT8 (11.2M weights), a separate GPU benchmark measures 41.2 ms per full Ours-B recovery scan versus 87.8 ms for in-place ECC, consistent with the relative trend in Figure 6. Ordinary inference uses materialized projected weights without repeating projection: on CIFAR-10/ResNet-18/INT8, with batch size 128 over 200 timing trials, Ours-B adds only 0.16% forward latency (18.65 ms versus 18.62 ms for None), compared with 1.34% for OCM (18.87 ms), excluding recovery scans. When targeted fault injection takes longer than a recovery scan, sufficiently frequent scans can also make coordinated multi-bit attacks harder to complete before correction; Appendix D discusses these timing requirements and evaluates stronger adaptive bit-flip attacks. Together, these results further demonstrate that our method combines negligible ordinary-inference overhead with substantially faster recovery than the evaluated ECC baseline, while total deployment cost depends on scan frequency and available memory bandwidth.

D Additional Adaptive Evaluation and Threat-Model Limits

Practical fault-injection constraints. Targeted fault injection faces substantial timing and locality constraints. DeepHammer [57] reports that earlier exploits could take days to induce one selected flip, while its optimized 2–24-bit attacks complete in minutes. It also selects one target per 4 KB page because multiple prescribed targets may be infeasible, making coordinated corruption within a 64-byte $K = 8$ INT8 block particularly challenging. These exploitation times far exceed our 41.2 ms full-model recovery scan, suggesting that periodic recovery can interrupt detectable corruption sequences if affected blocks are revisited between injections. Our main adaptive evaluation (Section 4)

deliberately grants greater attacker control by allowing repeated, recovery-aware targeting of the same block without these physical timing and locality restrictions.

Stronger detection-evading adaptive attacks. To accumulate damage despite periodic recovery, an attacker can seek flips that leave every intermediate state undetected. We therefore evaluate a threshold-evading search on CIFAR-10/ResNet-18/INT8 that accepts damaging payload-bit flips only when $\|r(Y)\|_\infty \le \varepsilon$, $\|c(Y)\|_\infty \le \varepsilon$, and all entries remain finite, using the deployed threshold that accounts for clean quantization residuals. As shown in Table 4, Ours-B retains 0.847 accuracy after 100 flips and approaches chance only at 800, whereas the undefended reference falls to 0.107 after five unrestricted flips. The hundreds of carefully constrained flips needed by this search, well beyond DeepHammer's demonstrated 2–24-bit budgets, further support substantial practical robustness against detection-evading bit-flip attacks.

Table 4: Accuracy under threshold-evading payload-bit corruption on CIFAR-10/ResNet-18/INT8.

Bit budget	0	5	10	50	100	200	300	400	800
Ours-B	0.912	0.911	0.910	0.900	0.847	0.633	0.441	0.142	0.098
None	0.908	0.107	0.107	0.107	0.107	0.107	0.107	0.107	0.107

Further limitations. An even stronger attacker could induce coordinated multi-bit flips simultaneously, or before recovery revisits the affected block, to construct residual-canceling patterns (Proposition 4). Our sequential searches do not exhaustively evaluate this capability; jointly optimized multi-bit attacks and their interaction with recovery scheduling remain future work.

E Matched Quantization Controls

Fixed-scale baseline comparison. To assess the contribution of recovery beyond protecting scale metadata, we retrain the baselines on CIFAR-10/ResNet-18/INT8 using the same fixed clipping bound $a = 2.576$ and quantization scale $\Delta = a/127$ as Ours-B. At cumulative BER $\rho = 10^{-4}$ ($10^{-2}\%$), matching the scale improves ECC's step-10 accuracy from 0.791 to 0.837, while Ours-B remains higher at 0.876 (Table 5). The tolerance-only baselines deteriorate under this shared range, indicating that imposing a common scale affects methods differently and is insufficient by itself. These results support the contribution of zero-sum detection and recovery beyond fixed-scale protection alone.

Table 5: Matched quantization control. Paired entries give native/fixed-scale accuracy; Ours-B uses the fixed scale in all three rows.

	Ours-B	None	WCAT	OCM	ECOC	ECC
Clean	0.912	0.908/0.911	0.912/0.911	0.909/0.916	0.905/0.914	0.899/0.911
Step 1	0.910	0.904/0.217	0.909/0.217	0.906/0.152	0.902/0.188	0.899/0.910
Step 10	0.876	0.748/0.104	0.772/0.104	0.619/0.093	0.608/0.089	0.791/0.837

Weight stability across clipping ranges. We train four independent Ours-B checkpoints on CIFAR-10/ResNet-18/INT8 with $a \in \{1, 2.576, 4, 6\}$ and $\Delta = a/127$, averaging weight statistics over the 21 protected layers. Across a sixfold increase in clipping range, the mean layer-wise standard deviation changes by only 6.6%, while the mean maximum weight magnitude remains between 0.194 and 0.203 (Table 6). These results demonstrate stable learned weight magnitudes across the tested nonbinding clipping ranges, despite substantial changes in the fixed quantization scale.

F Evaluation beyond Image Classification

Robustness in temporal classification and image reconstruction. We evaluate a four-layer LSTM [15] on ECG5000 [9] time-series classification and a U-Net [43] denoising autoencoder on CIFAR-10 reconstruction under cumulative corruption with per-bit probability $\rho = 10^{-4}$ ($10^{-2}\%$). Ours-B achieves the highest clean and corrupted LSTM macro-F1, retaining 0.570 at step 10. On

Table 6: Sensitivity of learned weight statistics to the fixed clipping range. The last column is mean layer maximum magnitude divided by a.

Bound a	Step Δ	Mean σ	Mean $\max\|W\|$	Range used
1.0	0.00787	0.0440	0.194	19.4%
2.576	0.02028	0.0446	0.200	7.8%
4.0	0.03150	0.0453	0.195	4.9%
6.0	0.04724	0.0469	0.203	3.4%

U-Net, despite slightly lower clean Peak Signal-to-Noise Ratio (PSNR), it retains 22.73 dB at step 10, exceeding ECC's 21.69 dB and WCAT's 19.90 dB (Table 7). These point-estimate comparisons extend the empirical support for our method to temporal inputs and dense reconstruction, demonstrating robustness beyond image classification in the evaluated settings.

Table 7: Cumulative-fault evaluation on sequential classification and dense image reconstruction. Higher values are better; the best value per column is **highlighted**.

Task / metric	Method	Clean	Step 1	Step 8	Step 10
LSTM / macro-F1	None	0.551	0.549	0.538	0.535
	WCAT	0.551	0.549	0.538	0.535
	In-place ECC	0.551	0.415	0.405	0.405
	Ours-B	**0.592**	**0.566**	**0.570**	**0.570**
U-Net / PSNR (dB)	None	**24.58**	23.39	21.31	19.89
	WCAT	**24.58**	23.92	21.26	19.90
	In-place ECC	24.53	21.60	21.64	21.69
	Ours-B	24.47	**23.92**	**22.98**	**22.73**

G Additional Training-Based Baseline Comparisons

We additionally train RQUANT+Clipping, RandBET, and AdvBET-FO [49] from scratch using the same training hyperparameters as the main experiments, and evaluate them under the exact ResNet-34/VGG-13 INT8 setting of Table 3. Table 8 reports the step-10 comparison.

H Additional Experiments and Diagnostics

How do projection and block construction affect performance? On CIFAR-10/ResNet-18/INT8, hard balanced projection with flattened blocks achieves 0.912 clean accuracy, compared with 0.909 for a soft zero-sum penalty ($\lambda = 1$) and 0.908 for hard projection with tensor-structured grouping $[\mathrm{out_channel}, \mathrm{in_channel}\, k_h k_w]$. Hard projection enforces the invariant before quantization without a penalty coefficient, whereas soft regularization leaves larger clean residuals and requires a looser detection threshold. Flattening also provides a reproducible block ordering without architecture-specific grouping rules. These observations support hard projection with flattened blocks as an effective default in this setting.

Does the structural constraint preserve clean performance? The zero-sum constraint retains $(K-1)^2/K^2$ of the unconstrained weight degrees of freedom, or $49/64 = 76.6\%$ at $K = 8$, without additional weight storage. The in-place SECDED baseline retains $57/64 = 89.1\%$ payload bits, a different measure of representational cost. On CIFAR-10/ResNet-18/INT8, clean accuracy is 0.912 for Ours-B, 0.908 for None, and 0.899 for ECC. Across ten matched architecture–precision settings, the mean clean-accuracy reduction is only 0.14 percentage points, with a maximum of 1.41 points for ViT-B/16-INT8. The mean layer-wise weight standard deviation across ResNet-18's 21 protected layers is 0.0481, within the 0.0461–0.0496 range of two independently trained undefended models. These results support a small average clean-performance cost while maintaining a comparable aggregate weight spread.

Table 8: Robust accuracy after 10 cumulative random corruption steps in INT8 at BER $10^{-2}\%$ ($\rho = 10^{-4}$ per step). Values are mean $\pm$ standard deviation over three runs; the highest mean per column is **highlighted**.

ResNet-34					
Method	CIFAR10	CIFAR100	GTSRB	ImageNet1K	SVHN
None	$0.8105_{\pm 0.1650}$	$0.5921_{\pm 0.1213}$	$0.9076_{\pm 0.0744}$	$0.0931_{\pm 0.0852}$	$0.9548_{\pm 0.0024}$
WCAT	$0.8298_{\pm 0.1301}$	$0.5921_{\pm 0.1213}$	$0.9476_{\pm 0.0536}$	$0.1245_{\pm 0.1108}$	$0.9528_{\pm 0.0007}$
OCM	$0.8078_{\pm 0.1689}$	$0.5223_{\pm 0.2766}$	$0.9577_{\pm 0.0446}$	$0.1105_{\pm 0.1007}$	$0.9583_{\pm 0.0059}$
ECOC	$0.7179_{\pm 0.3351}$	$0.5630_{\pm 0.2013}$	$0.9517_{\pm 0.0444}$	$0.1489_{\pm 0.1318}$	$0.9604_{\pm 0.0045}$
ECC	$0.8083_{\pm 0.1744}$	$0.5441_{\pm 0.2158}$	$0.9668_{\pm 0.0255}$	$0.4103_{\pm 0.3532}$	$0.9552_{\pm 0.0018}$
RQUANT+Clipping	$0.8345_{\pm 0.1125}$	$0.5190_{\pm 0.2593}$	$0.9588_{\pm 0.0197}$	$0.1004_{\pm 0.0884}$	$0.9601_{\pm 0.0041}$
RandBET	$0.8531_{\pm 0.0241}$	$0.6059_{\pm 0.0412}$	$0.9683_{\pm 0.0040}$	$0.1203_{\pm 0.0905}$	$0.9561_{\pm 0.0027}$
AdvBET-FO	$0.8383_{\pm 0.0676}$	$0.5762_{\pm 0.1868}$	$0.9608_{\pm 0.0038}$	$0.1003_{\pm 0.0905}$	$0.9609_{\pm 0.0008}$
Ours-U	$0.8644_{\pm 0.0107}$	$0.5518_{\pm 0.0339}$	$0.9388_{\pm 0.0178}$	$0.4904_{\pm 0.0340}$	$0.9499_{\pm 0.0020}$
Ours-B	$\mathbf{0.9151}_{\pm 0.0019}$	$\mathbf{0.6255}_{\pm 0.0132}$	$\mathbf{0.9747}_{\pm 0.0014}$	$\mathbf{0.5098}_{\pm 0.0236}$	$\mathbf{0.9749}_{\pm 0.0011}$
VGG-13					
Method	CIFAR10	CIFAR100	GTSRB	ImageNet1K	SVHN
None	$0.9062_{\pm 0.0013}$	$0.6880_{\pm 0.0051}$	$0.9696_{\pm 0.0190}$	$0.2633_{\pm 0.0880}$	$0.9565_{\pm 0.0015}$
WCAT	$0.9155_{\pm 0.0018}$	$0.6913_{\pm 0.0066}$	$0.8748_{\pm 0.1530}$	$0.2399_{\pm 0.0714}$	$0.9556_{\pm 0.0030}$
OCM	$0.9090_{\pm 0.0066}$	$0.6608_{\pm 0.0066}$	$0.9847_{\pm 0.0007}$	$0.4650_{\pm 0.0361}$	$0.9619_{\pm 0.0021}$
ECOC	$0.9072_{\pm 0.0034}$	$0.6738_{\pm 0.0013}$	$0.9757_{\pm 0.0205}$	$0.4551_{\pm 0.0454}$	$0.9644_{\pm 0.0018}$
ECC	$0.9031_{\pm 0.0003}$	$0.6873_{\pm 0.0022}$	$0.7910_{\pm 0.3201}$	$0.4508_{\pm 0.0003}$	$0.9565_{\pm 0.0000}$
RQUANT+Clipping	$0.9136_{\pm 0.0045}$	$0.6707_{\pm 0.0022}$	$0.9821_{\pm 0.0064}$	$0.2370_{\pm 0.0962}$	$0.9631_{\pm 0.0016}$
RandBET	$0.9053_{\pm 0.0027}$	$0.6929_{\pm 0.0013}$	$0.9832_{\pm 0.0012}$	$0.2159_{\pm 0.0728}$	$0.9645_{\pm 0.0014}$
AdvBET-FO	$0.8883_{\pm 0.0034}$	$0.6152_{\pm 0.0080}$	$0.9585_{\pm 0.0042}$	$0.2173_{\pm 0.0626}$	$0.9561_{\pm 0.0013}$
Ours-U	$0.6413_{\pm 0.3997}$	$0.4674_{\pm 0.1714}$	$0.8089_{\pm 0.1892}$	$0.0012_{\pm 0.0003}$	$0.8035_{\pm 0.2002}$
Ours-B	$\mathbf{0.9230}_{\pm 0.0015}$	$\mathbf{0.6946}_{\pm 0.0081}$	$\mathbf{0.9992}_{\pm 0.0004}$	$\mathbf{0.4651}_{\pm 0.0381}$	$\mathbf{0.9809}_{\pm 0.0011}$

How does our method affect training cost and local loss sensitivity? On CIFAR-10/ResNet-18/INT8, epoch-time ratios relative to natural training are $1.01\times$ for Ours-B, $1.03\times$ for WCAT, $1.97\times$ for OCM, $2.36\times$ for ECOC, and $2.54\times$ for in-place ECC under the common training schedule. We measure local sensitivity using the gradient-direction proxy $S_\eta = \mathcal{L}(w + \eta g/\|g\|_2) - \mathcal{L}(w)$, where $g = \nabla_w \mathcal{L}(w)$. As shown in Table 9, Ours-B nearly matches natural training at the smallest radius (0.083 versus 0.082), exceeds it at larger radii, and remains less sensitive than ECC at all tested radii. Together with its lowest observed clean loss, these results demonstrate low per-epoch training overhead and effective clean-data fitting under the structural constraint.

Table 9: Clean loss and first-order sharpness on CIFAR-10/ResNet-18/INT8.

Method	Clean loss	$S_{.01}$	$S_{.05}$	$S_{.1}$
None	0.389	0.082	0.550	2.003
WCAT	0.355	0.082	0.615	2.159
In-place ECC	0.501	0.109	1.036	3.815
Ours-B	0.333	0.083	0.715	2.763

How accurately does our method recover corrupted weights? At cumulative step 10 on CIFAR-10/ResNet-18/INT8, we evaluate approximately 9,000 corrupted weights using bit-exact agreement and numerical error $d_i = s|\widehat{q}_i - c_i|$, where c_i is the clean integer weight, $\widehat{q}_i$ its recovered value, and s the uncorrupted scale. Despite lower bit-exact agreement, Ours-B achieves smaller numerical errors than ECC at both evaluated BERs, reducing mean and 90th-percentile error by approximately 42% and 32%, respectively, at BER 10^{-4} (Table 10). No false corrections are observed on the clean models evaluated with the calibrated maximum-clean-residual threshold. These findings demonstrate that Ours-B can better preserve numerical weight values despite less frequent bit-exact reconstruction.

I Theoretical Analysis

This section formalizes the algebraic structure induced by the proposed zero-sum projection, derives recovery guarantees under random corruption, characterizes undetectable error patterns, and compares the resulting guarantees with SECDED-style ECC. The analysis is intended to explain the main empirical trends reported in Section 5 and Appendix: strong robustness under cumulative random

Table 10: Recovery diagnostics conditional on corrupted weights. BER is a probability per bit (not a percentage); p90 is the 90th percentile of d_i.

BER	Method	Bit-exact	Mean d_i	p90 d_i
5×10^{-5}	Ours-B	26.8%	0.048	0.088
	In-place ECC	99.7%	0.069	0.134
10^{-4}	Ours-B	26.8%	0.053	0.122
	In-place ECC	99.5%	0.092	0.180

corruption, the dependence on block size K, and the greater difficulty of mounting adaptive attacks against a distributed recovery structure than against concentrated ECC code blocks.

I.1 Preliminaries

For a square block of side length q, let $\mathbf{1}_q \in \mathbb{R}^q$ denote the all-ones vector and define the zero-sum subspace

$$\mathcal{S}_q = \left\{ X \in \mathbb{R}^{q \times q} : \; X\mathbf{1}_q = \mathbf{0}, \; \mathbf{1}_q^\top X = \mathbf{0}^\top \right\}. \tag{20}$$

For a clean block $G \in \mathcal{S}_q$, the deployed corrupted block is

$$Y = G + E, \tag{21}$$

where $E \in \mathbb{R}^{q \times q}$ is an unknown corruption matrix.

We define the row and column syndrome vectors

$$r(Y) = Y\mathbf{1}_q, \qquad c(Y) = Y^\top \mathbf{1}_q. \tag{22}$$

Because $G \in \mathcal{S}_q$, these depend only on the corruption:

$$r(Y) = E\mathbf{1}_q, \qquad c(Y) = E^\top \mathbf{1}_q. \tag{23}$$

Thus the recovery signal used by Algorithm 2 is exactly the row- and column-sum signature of the corruption. This directly matches the recovery procedure in the main method section, where detection and correction are driven by row and column residuals. The exact-recovery statements below assume an exact zero-sum clean block and exact arithmetic; they do not assert bit-exact recovery after finite-precision mapping. Deployed thresholds and numerical recovery are evaluated separately in Appendix H.

I.2 Balanced projection as an orthogonal projector

Let

$$H_q = I_q - \frac{1}{q}\mathbf{1}_q\mathbf{1}_q^\top \tag{24}$$

be the centering matrix. The balanced projection in the main text can be written compactly as

$$\Pi_q(X) = H_q X H_q. \tag{25}$$

Proposition 1 (Balanced zero-sum projection)**.** *For every $X \in \mathbb{R}^{q \times q}$, $\Pi_q(X) \in \mathcal{S}_q$. Moreover, Π_q is the orthogonal projector onto $\mathcal{S}_q$ under the Frobenius inner product.*

Proof. Since $H_q\mathbf{1}_q = \mathbf{0}$ and $\mathbf{1}_q^\top H_q = \mathbf{0}^\top$,

$$\Pi_q(X)\mathbf{1}_q = H_q X H_q \mathbf{1}_q = \mathbf{0}, \qquad \mathbf{1}_q^\top \Pi_q(X) = \mathbf{1}_q^\top H_q X H_q = \mathbf{0}^\top, \tag{26}$$

so $\Pi_q(X) \in \mathcal{S}_q$. Since H_q is symmetric and idempotent, Π_q is also symmetric and idempotent with respect to the Frobenius inner product, hence it is the orthogonal projector onto $\mathcal{S}_q$. □

This proposition formalizes why the balanced variant preserves parameter geometry more naturally than the unbalanced chart-based parameterization: it is the Euclidean projection onto the target constraint set, rather than a coordinate-dependent reparameterization.

I.3 Exact recovery for isolated corruption

The simplest regime is a single corrupted entry.

Theorem 1 (Exact localization and correction of one corrupted entry)**.** *Let $G \in \mathcal{S}_q$ and suppose*

$$Y = G + \delta e_i e_j^\top, \qquad \delta \neq 0, \tag{27}$$

for some $i, j \in \{1, \dots, q\}$. Then

$$r(Y) = \delta e_i, \qquad c(Y) = \delta e_j. \tag{28}$$

Hence the corrupted coordinate (i, j) and magnitude δ are uniquely identified, and

$$G = Y - \delta e_i e_j^\top. \tag{29}$$

Proof. Using Equation 23,

$$r(Y) = \delta e_i e_j^\top \mathbf{1}_q = \delta e_i, \qquad c(Y) = \delta e_j e_i^\top \mathbf{1}_q = \delta e_j. \tag{30}$$

Thus the unique nonzero row syndrome identifies i, the unique nonzero column syndrome identifies j, and their common value is δ. □

Theorem 1 explains the strong performance under sparse random corruption observed in the cumulative random-fault experiments: whenever a block remains in the single-entry regime, the exact-elimination phase of Algorithm 2 recovers it perfectly.

I.4 A sufficient recovery guarantee under random corruption

We now analyze the standard independent corruption model. Consider a regular $K \times K$ block, so that the protected unit contains

$$n = K^2 \tag{31}$$

entries. Assume each entry is corrupted independently with probability ρ.

Theorem 1 gives an exact guarantee whenever the block has at most one corrupted entry. This is a sufficient condition for exact recovery; the algorithm may still recover some multi-entry patterns, but we do not rely on that here.

Theorem 2 (Lower bound on block exact-recovery probability)**.** *Under independent entry corruption rate ρ, the exact recovery probability of a $K \times K$ regular block satisfies*

$$P_{\mathrm{rec}}^{\mathrm{ZS}}(K, \rho) \;\geq\; (1-\rho)^{K^2} + K^2 \rho (1-\rho)^{K^2-1}. \tag{32}$$

Equivalently, the exact recovery failure probability satisfies

$$P_{\mathrm{fail}}^{\mathrm{ZS}}(K, \rho) \;\leq\; 1 - (1-\rho)^{K^2} - K^2 \rho (1-\rho)^{K^2-1}. \tag{33}$$

For small ρ,

$$P_{\mathrm{fail}}^{\mathrm{ZS}}(K, \rho) \;\leq\; \binom{K^2}{2} \rho^2 + O(\rho^3). \tag{34}$$

Proof. The events "zero corrupted entries" and "exactly one corrupted entry" imply exact recovery by Theorem 1. Under independent Bernoulli corruption over K^2 entries, these events have probabilities $(1-\rho)^{K^2}$ and $K^2\rho(1-\rho)^{K^2-1}$, respectively. Summing them gives the lower bound Equation 32. The expansion Equation 34 follows from Taylor expansion. □

Thus, under standard random corruption, exact block failure is at most second order in ρ. The leading coefficient grows like $\Theta(K^4)$, which helps explain the block-size ablation: for fixed corruption rate, increasing K enlarges the protected unit and therefore eventually weakens the exact sparse-recovery regime, even though larger K may improve clean performance. This is consistent with the empirical observation that moderate K, particularly near $K = 8$, yields the strongest overall robustness–performance balance.

Bit-level corruption model. In quantized deployment, corruption is injected at the bit level. If each weight uses b bits and each bit flips independently with rate p, then the induced entry-corruption rate is

$$\rho = 1 - (1-p)^b. \tag{35}$$

Substituting Equation 35 into Equation 32 yields the induced block-level lower bound under bit flips:

$$P_{\mathrm{rec}}^{\mathrm{ZS}}(K,p) \;\geq\; (1-\rho)^{K^2} + K^2\rho(1-\rho)^{K^2-1}, \qquad \rho = 1-(1-p)^b. \tag{36}$$

Corollary 1 (Small-bit-error approximation)**.** *For small p,*

$$\rho = bp + O(p^2), \tag{37}$$

and therefore

$$P_{\mathrm{fail}}^{\mathrm{ZS}}(K,p) \;\leq\; \binom{K^2}{2} b^2p^2 + O(p^3). \tag{38}$$

I.5 Layer-level guarantee

Suppose a layer consists of M independent regular blocks, each corrupted under the same per-entry rate ρ. Then the probability that all blocks satisfy the one-entry sufficient condition is

$$\left((1-\rho)^{K^2} + K^2\rho(1-\rho)^{K^2-1}\right)^M, \tag{39}$$

which implies the lower bound

$$P_{\mathrm{layer}}^{\mathrm{ZS}} \;\geq\; \left((1-\rho)^{K^2} + K^2\rho(1-\rho)^{K^2-1}\right)^M. \tag{40}$$

Equivalently, by the union bound,

$$P_{\mathrm{layer\ fail}}^{\mathrm{ZS}} \;\leq\; M\left[1 - (1-\rho)^{K^2} - K^2\rho(1-\rho)^{K^2-1}\right]. \tag{41}$$

Although conservative, Equation 41 already shows why cumulative corruption can remain manageable when the per-step error rate is small and recovery is applied repeatedly: the blockwise failure probability remains second order in the corruption rate, while exact single-entry correction is applied after every step.

I.6 Iteration complexity of the exact-elimination phase

Algorithm 2 first performs exact elimination whenever a row or column isolates a unique candidate entry, and only then applies a soft correction to the remaining candidate set. The exact-elimination phase has a simple worst-case bound.

Proposition 2 (Worst-case exact-elimination complexity)**.** *For a $q \times q$ block, the exact-elimination phase of Algorithm 2 terminates after at most q^2 elimination steps.*

Proof. Each elimination step resolves at least one previously unresolved candidate location. Since a $q \times q$ block contains at most q^2 entries, no more than q^2 eliminations are possible. □

In practice the number of elimination steps is much smaller. Single-entry corruption requires one exact update, and sparse multi-entry corruption requires at most the number of entries that become uniquely identifiable during peeling. Because the row/column syndromes, candidate masks, and updates are implemented by reductions, broadcasts, and masked tensor operations, the layer-level recovery cost scales linearly with the number of stored weights and remains well aligned with dense accelerator execution. This matches the runtime measurements in Appendix C, where our method exhibits substantially lower recovery overhead than ECC.

I.7 Undetectable error patterns

The exact stealth condition is characterized by the same zero-sum subspace.

Proposition 3 (Undetectable corruption)**.** *A corruption matrix $E \in \mathbb{R}^{q\times q}$ is perfectly undetectable by row/column checksum testing if and only if*

$$E\mathbf{1}_q = \mathbf{0}, \qquad \mathbf{1}_q^\top E = \mathbf{0}^\top, \tag{42}$$

that is, if and only if $E \in \mathcal{S}_q$.

Proof. By Equation 23, the observed syndromes are $r(Y) = E\mathbf{1}_q$ and $c(Y) = E^\top \mathbf{1}_q$. Both vanish exactly when $E \in \mathcal{S}_q$. □

Under exact arithmetic and zero detection threshold, an isolated nonzero corruption cannot be syndrome-invisible. With finite precision and $\varepsilon > 0$, however, even isolated changes can remain below the deployed threshold; Appendix D evaluates this case.

Proposition 4 (Minimum support of a nonzero undetectable corruption)**.** *Let $E \neq 0$ satisfy $E \in \mathcal{S}_q$. Then $\mathrm{supp}(E)$ has cardinality at least 4. Moreover, this bound is tight.*

Proof. A support of size 1 changes one row sum and one column sum, so it cannot lie in $\mathcal{S}_q$. A support of size 2 or 3 cannot make all affected row sums and column sums vanish simultaneously: every affected row must contain at least two cancelling contributions, and every affected column must also contain at least two cancelling contributions, which is impossible with fewer than four nonzero locations. Tightness is achieved by any embedded 2×2 checkerboard pattern

$$E = \delta \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \qquad \delta \neq 0,$$

whose row and column sums are all zero. □

Proposition 4 provides a useful structural contrast with ECC. For our method, exact stealth requires at least a coordinated four-entry pattern inside a block, whereas standard SECDED-style ECC already loses exact correction after only two bit errors in a codeword.

I.8 Implications for the empirical results

The analysis above explains three recurring empirical patterns in the main text.

First, the cumulative random-fault results are consistent with Theorem 2: exact failure under random corruption is at most second order in the per-entry corruption rate, which explains why the proposed method remains highly stable when corruption is injected incrementally and recovery is applied after each step.

Second, the block-size ablation in Appendix B is consistent with the coefficient $\binom{K^2}{2}$ in Equation 34: for fixed corruption rate, the exact sparse-recovery regime becomes less favorable as K grows, which naturally leads to an intermediate optimal block size rather than monotone improvement with increasing K.

Third, Proposition 4 and the ECC comparison clarify the adaptive-attack results. Exact stealth against the proposed method requires a coordinated multi-entry pattern inside a block, whereas ECC already loses exact correction after two bit errors in a codeword. This structural difference helps explain why ECC is more readily neutralized by a small number of carefully chosen flips, while the proposed method exhibits a more distributed vulnerability surface.

J Additional Figures

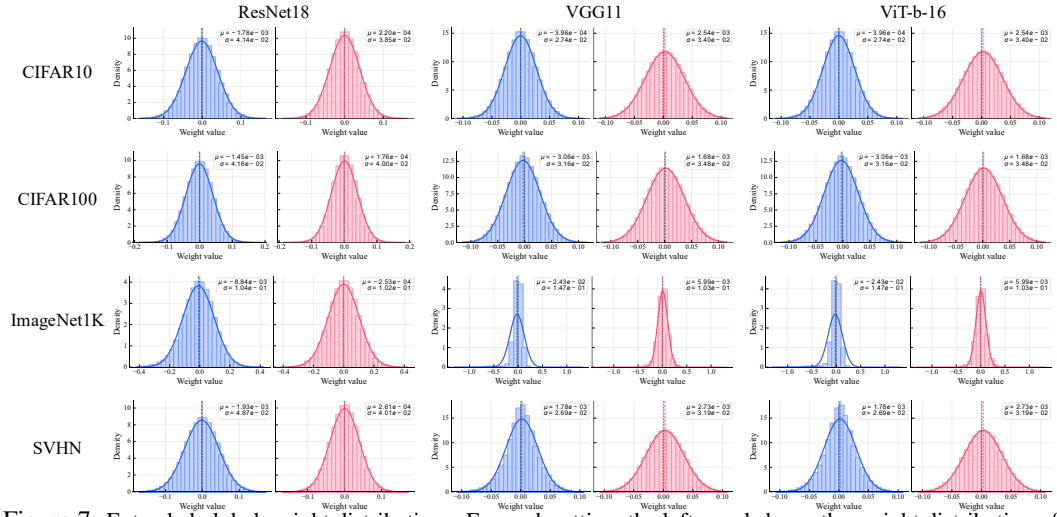

Figure 7: Extended global weight distributions. For each setting, the left panel shows the weight distribution of the naturally trained model, and the right panel shows the distribution of the model trained with our balanced projection method using block size $K = 8$. The two distributions remain highly similar in overall shape, while our method produces weights that are slightly more concentrated around zero.

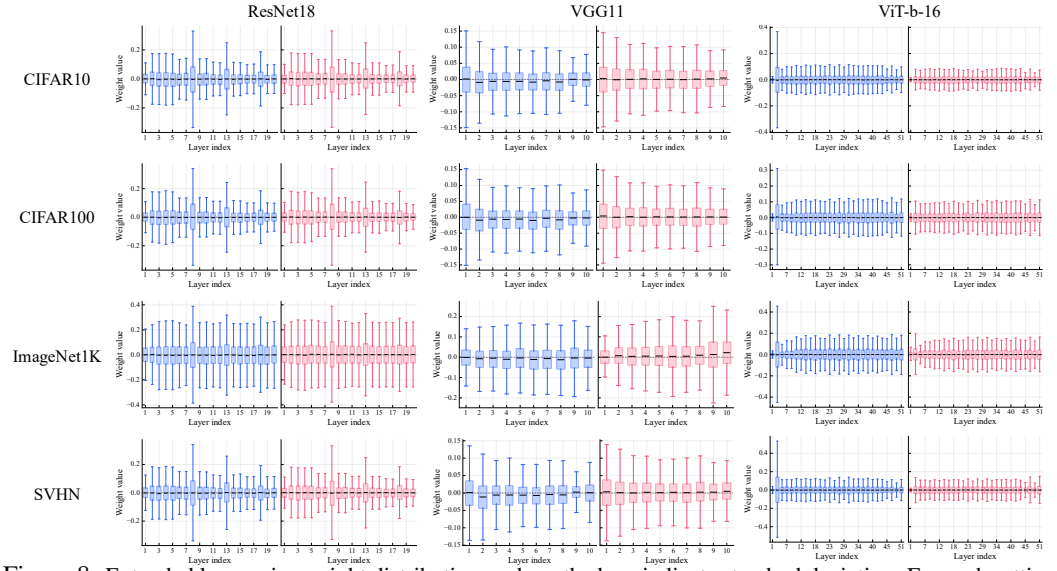

Figure 8: Extended layer-wise weight distributions, where the bars indicate standard deviation. For each setting, the left panel shows the naturally trained model and the right panel shows the model trained with our balanced projection method using block size $K = 8$. The layer-wise trends remain highly similar after projection-based training, indicating that our method aligns well with the model's natural weight structure while only minimally perturbing training. The comparable scale range across layers also supports our use of a fixed quantization scale to avoid introducing additional vulnerability through mutable scale factors.

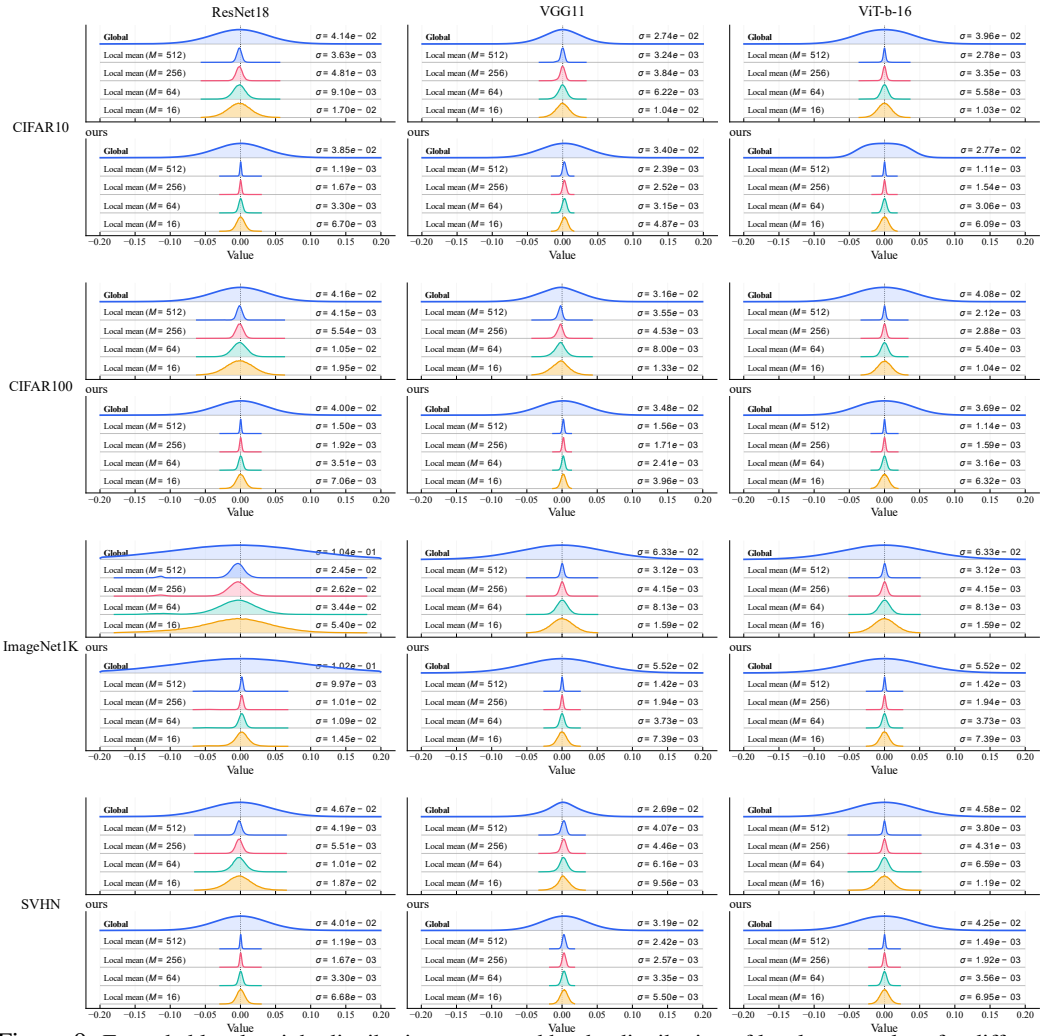

Figure 9: Extended local weight distributions measured by the distribution of local mean values for different local sizes M, where M denotes the number of consecutive flattened weights in each local group. For each setting, the top panel shows the naturally trained model and the bottom panel shows the model trained with our balanced projection method using block size $K = 8$. Naturally trained models already exhibit local means concentrated near zero, and our method further sharpens this behavior by enforcing more tightly zero-centered local structure.

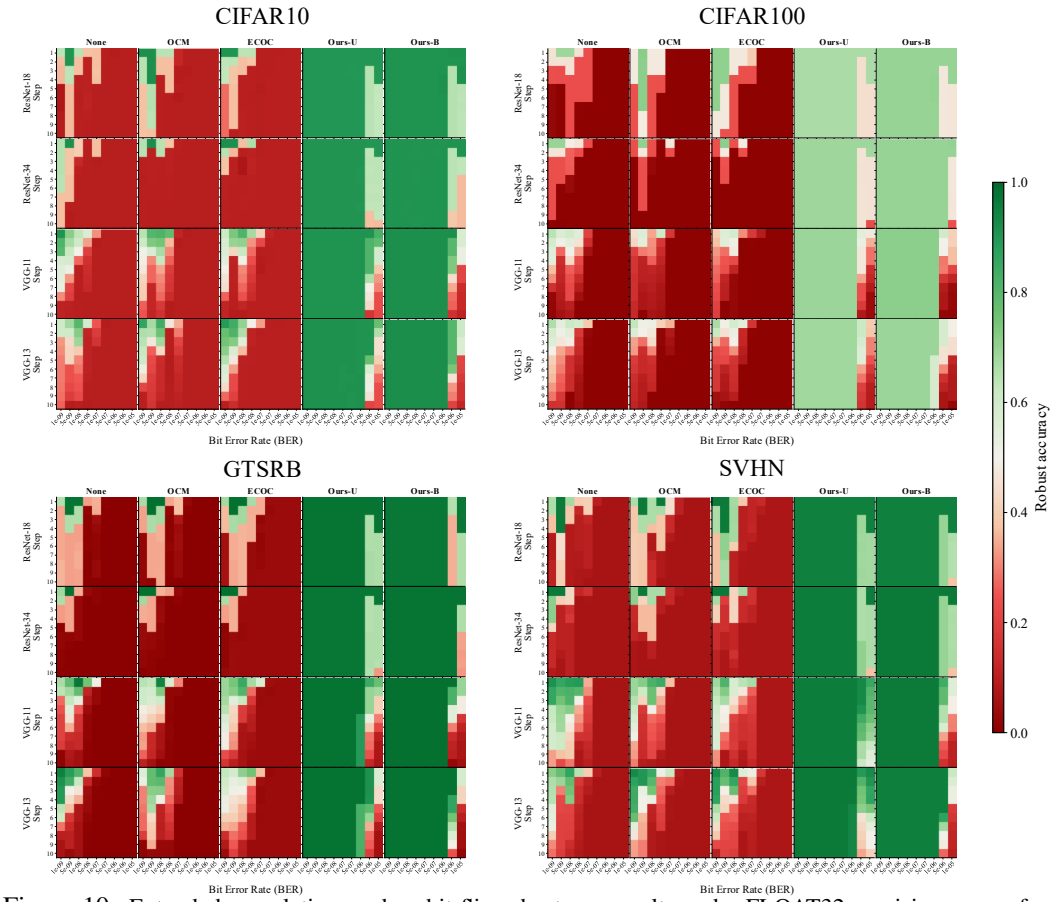

Figure 10: Extended cumulative random bit-flip robustness results under FLOAT32 precision across four datasets (CIFAR-10, CIFAR-100, GTSRB, and SVHN) and four target models (ResNet-18, ResNet-34, VGG-11, and VGG-13). Each heatmap reports robust accuracy averaged over three runs across cumulative recovery steps $\{1, 2, \ldots, 10\}$ and BERs $\{10^{-9}, 5 \times 10^{-9}, 10^{-8}, 5 \times 10^{-8}, \ldots, 10^{-5}\}$. While competing defenses degrade rapidly at moderate BERs or after a small number of cumulative steps, our method remains substantially more stable across both axes, showing markedly stronger robustness to accumulated bit-flip noise.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the paper's main contributions: a training-based recovery framework using block-wise zero-sum constraints, adaptive attacks against recovery-based defenses, and broad empirical evaluation across architectures, precisions, datasets, and threat models. These claims are aligned with the methodological, theoretical, and experimental results reported in the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a paragraph in the conclusion, noting that the current method uses simple flattening-based block construction and that more effective block construction and block-size selection remain open problems.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The paper states assumptions for its theoretical claims in the appendix and defers the full formal analysis, including proofs and comparison with ECC-based recovery. Theorems, propositions, and assumptions are explicitly stated and cross-referenced.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper specifies datasets, models, precisions, baselines, attack methods, main hyperparameters, evaluation metrics, and threat evaluation protocols in sufficient detail to understand and reproduce the main empirical results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The supplemental material provides organized project code. The experiments rely on standard public datasets and commonly used model architectures.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the datasets, architectures, training duration, precisions, quantization setting, block size, code lengths for OCM-based methods, attack budgets, BER settings, and evaluation procedures. These details are sufficient to understand the experimental setup, with additional implementation details provided in the supplementary material.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: The paper reports mean $\pm$ standard deviation over three runs in tables and discusses averaged results in captions and text. The source of variability is clearly the repeated runs under the same experimental setting.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies the main hardware platform used for experiments (an NVIDIA H200 GPU with 150GB memory).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work is methodological and experimental, uses standard benchmark datasets and models, does not involve human subjects, and does not appear to violate the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: The paper does not include a dedicated broader impacts discussion. The work is primarily methodological.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk generative models or scraped datasets, and the released assets are training code for robustness research rather than models with obvious misuse risk of the type described in the checklist.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites the benchmark datasets, models, attacks, and baseline methods used.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper does not release new datasets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are not part of the core method, theory, or experiments described in the paper.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.