SINAI: Selective Injection of Noise with Architectural Integration for Robust and Efficient Edge Vision AI

Zhenyu Liu
Rensselaer Polytechnic Institute
USA

Yayue Hou
Rensselaer Polytechnic Institute
USA

Garrett Gagnon
Rensselaer Polytechnic Institute
USA

Sanchari Sen
IBM Research
USA

Swagath Venkatarmani
IBM Research
USA

Nan Wu
George Washington University
USA

Liu Liu
Rensselaer Polytechnic Institute
USA

Abstract

Deploying Deep Neural Networks (DNNs) on edge platforms presents two major challenges: the rising demand for execution efficiency as models scale, and the need for robustness against adversarial attacks. While modern hardware accelerators emphasize performance and energy efficiency, they often overlook robustness. In contrast, existing algorithmic defenses improve robustness but incur additional operations, compromising the deployment on resource-constrained hardware. To bridge this gap, we propose SINAI: a hardware-aware co-design framework with selective injection of noise with architectural integration to enhance adversarial robustness and execution efficiency. Based on the key insight that only a subset of activations forming critical pathways are important to model accuracy, we propose to identify these critical pathways and inject noise selectively into non-critical pathways. At the architecture level, SINAI extends a generic neural processing unit (NPU) with lightweight support for multi-precision computation, threshold-based selection, and activation sparsity, enabling efficient integration without intrusive hardware changes. We further introduce an enhanced design that aligns noise injection granularity with hardware-friendly structured patterns. Experimental results across multiple DNNs and datasets show that our work achieves up to 15.24% higher robust accuracy under adversarial attack, while delivering up to 7.8× speedup and 5.56× energy efficiency gains over baseline architectures.

1 Introduction

Deep Neural Networks (DNNs) are at the forefront of technological advancements, enabling a wide range of intelligent applications across various sectors [36]. Yet, deploying DNNs on edge devices or resource-constrained platforms introduces significant challenges. On the one hand, the limited resources of such devices are often insufficient to handle the increasing complexity of modern DNNs [48, 54]. On the other hand, their vulnerability to adversarial attacks poses serious concerns in mission-critical tasks [4]. For instance, an edge-based self-driving system might misinterpret traffic signs or a security device could fail in threat detection due to adversarial examples—images with subtle, invisible changes that significantly disrupt classification accuracy [8, 34, 45]. Therefore, techniques that enhance both the efficiency and robustness of DNNs are essential for deployment on edge devices.

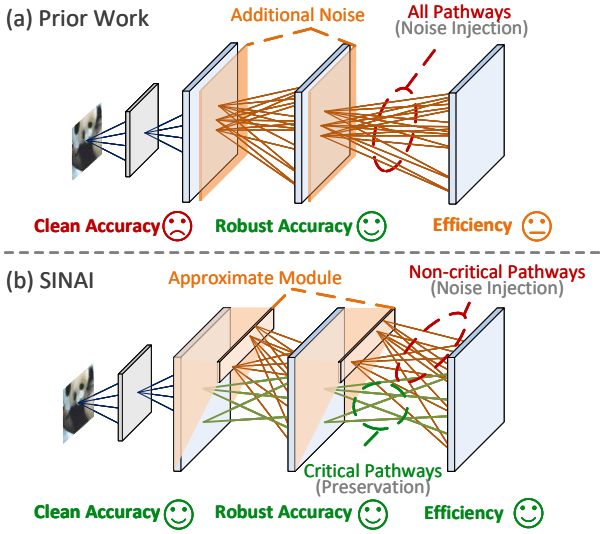

Figure 1: Prior work on noise injection vs. SINAI. (a) Prior work adds noise to all pathways, reducing clean accuracy and ignoring efficiency. (b) SINAI adds noise only to non-critical pathways, keeping accuracy and improving efficiency.

To address the first challenge, numerous accelerators have been proposed to enhance compute efficiency. Among these, sparse architectures [26, 47, 63, 67] have demonstrated near-lossless model accuracy by effectively exploiting sparsity. Meanwhile, from the algorithm level, various methods [12, 15, 25, 28, 50, 52, 62] have been introduced to defend DNNs against the adversarial attack challenge, showing potentials in mitigating robustness concerns. One notable approach is noise injection [12, 25, 28, 44, 50, 62], which has been shown to enhance robustness in both practice and theory [50].

While much effort has been dedicated to improving either efficiency or robustness, few approaches successfully address both challenges in a unified and hardware-friendly manner. On the one hand, sparsity-based methods offer only limited improvements in adversarial robustness: moderate sparsity yields marginal gains, while overly sparse networks may become even more vulnerable [23]. On the other hand, although noise injection is orthogonal to efficient methods and can be integrated into most models, it often

leads to an unavoidable drop in clean accuracy, making it less suitable for practical deployment. Moreover, existing architectures that aim to be both robust and efficient [15, 21] often suffer from slow convergence or require intrusive hardware modifications, which hinders their widespread adoption.

Given the limitations in current methods, we pose a question: *Is it possible to combine the advantages of sparsity-oriented accelerators and noise injection to achieve enhancements on both efficiency and robustness?* If successful, such an architectural integration could offer a new opportunity: by designing an effective noise injection strategy that is compatible with existing sparse architectures, we can potentially enhance adversarial robustness while maintaining model accuracy and execution efficiency without intrusive hardware designs. We term this approach – *Selective Injection of Noise with Architectural Integration (SINAI).*

Specifically, we draw inspiration from sparsity methods that treat activations differently and observe that activations exhibit various sensitivities to noise. Some activations are essential for maintaining clean accuracy and are highly sensitive to noise injection. We refer to the connection of essential activations as the Critical Pathway. Others can be injected noise without performance loss. Thus, we hypothesize that selectively introducing noise into non-critical pathways could enhance adversarial robustness without degrading clean accuracy. Motivated by this, we propose a novel method, named *pathway-aware selective noise injection*, to identify and inject noise only to non-critical pathways and preserve critical information propagation from critical pathways. As shown in Figure 1(a), prior work uniformly apply noise injection on both critical and non-critical pathways – compromising clean accuracy. Instead, Figure 1(b) illustrates our method with selective injection of noise only into non-critical pathways can maintain clean and robust accuracy.

Furthermore, we explore the design considerations of pathway-aware selective noise injection on generic NPU architectures. **1) Dynamic Identification of Critical Pathways:** The first key step in our method is to determine which pathways are critical for preserving clean accuracy. Since critical pathways are input-dependent and dynamically changing with each sample, they cannot be predefined or statically encoded. Therefore, an efficient online predictor is required to identify critical and non-critical pathways in runtime during inference. **2) Architectural Integration of Selective Noise Injection:** The second step is to ensure that the proposed noise injection method is hardware-efficient and does not require intrusive modifications to existing architectures.

To this end, we propose SINAI: Selective Injection of Noise with Architectural Integration for robust and efficient DNN inference. SINAI addresses the limitations of prior adversarial defense methods by unifying lightweight, learnable approximation with hardware-aware execution. At the algorithm level, SINAI leverages a trainable predictor to identify critical pathways and selectively inject noise by approximating non-critical activations. At the hardware level, SINAI introduces architectural support for selective noise injection on a generic sparse NPU, exploiting structured sparsity patterns and low-precision computation to enhance execution efficiency. Our key contributions are summarized as follows:

Table 1: Summary of attacks used in evaluation.

Type	Method	Strength	Norm	Core Idea
Black Box	Square [5]	**	ℓ_∞	Gradient-free, randomized search under query budget.
White Box	FGSM [20]	*	ℓ_∞	One-step gradient sign method.
	MI-FGSM [13]	**	ℓ_∞	FGSM + momentum for stability.
	PGD [46]	***	ℓ_∞	Iterative gradient ascent with projection.
	AutoAttack [11]	***	ℓ_∞	Combined parameter-free strong attacks.
	CW [8]	***	ℓ_2	Optimizes confidence-based loss.
	EAD [9]	***	ℓ_1/ℓ_2	Sparse elastic-net based optimization.

- **Pathway-Aware Selective Noise Injection** (Section 4): We present a novel method that bridges the benefits of noise injection and activation sparsity. By learning to identify critical pathways during inference, SINAI injects noise via approximation in non-critical pathways, improving adversarial robustness without sacrificing clean accuracy.
- **Architectural Integration on Specialized NPUs** (Section 5): We study the architectural integration of pathway-aware noise injection into generic NPUs specialized with multi-precision, threshold-based selection, and activation sparsity.
- **Structured Noise Injection** (Section 6): We extend our baseline architecture with SINAI-e, an enhanced variant of SINAI that employs structured noise injection to improve execution regularity. By enforcing structured selection of critical pathways, SINAI-e addresses inefficiencies in weight reuse and writeback caused by unstructured selection.
- We evaluate our algorithm-architecture co-design framework on four DNNs and three datasets under different attack scenarios in Section 7. We demonstrate that our framework, SINAI, achieves up to 7.8 × speedup and 5.56 × better energy efficiency over a generic NPU baseline and up to 15.24% robust accuracy gain under AutoAttack [11] without clean accuracy loss compared with Overfitting Adversarial Training [52].

2 Background

In this section, we introduce the background of adversarial attacks on DNNs. Then, we analyze the limitations of common defense methods based on noise injection. Finally, we introduce Specialized NPU Architectures designed to improve DNN efficiency.

2.1 Adversarial Robustness

Adversarial attacks have emerged as a significant threat to the deployment of DNNs [11]. Formally, given an input sample x with the true label y, an adversarial attack search for a small perturbation δ under an L_p-norm constraint that maximizes the model's loss function $\mathcal{L}$:

$$\delta = \arg \max_{\|\delta\|_p \leq \epsilon} \mathcal{L}(f_\theta(x+\delta), y) \tag{1}$$

where f_θ represents the DNN model parameterized by θ and ϵ controls the perturbation magnitude.

The measure of adversarial robustness builds upon the effectiveness of attacks, which quantifies the resilience of a model to such perturbations. By comparing the accuracy of a model on clean and

unperturbed data versus its accuracy on adversarial inputs, we can measure the robustness of a model [8]. Without defense, models can experience a sharp drop in accuracy – often more than 20% – even under simple attacks. In this paper, we consider both white and black scenarios as the attack models:

Black-box Attack. The attacker can only observe the model's input-output behavior, without access to its parameters or internal gradients. Typically, the attacker queries the model multiple times to approximate its behavior and then crafts adversarial examples based on these queries. A representative method used in this paper is the *Square Attack* [5], which relies on random search strategies.

White-box Attack. The attacker has full knowledge of the model's architecture, parameters, and defense mechanisms, enabling direct computation of gradients. Common gradient-based attack methods evaluated in this paper include *PGD* [46], *FGSM* [20], *MIFGSM* [13], *AutoAttack* [11], *CW* [8] and *EAD* [9]. These attacks generate stronger adversarial perturbations by directly optimizing the model's loss. Table 1 provides a detailed comparison and descriptions of the attack methods.

To enhance robustness against adversarial attacks, Adversarial Training (AT) is a foundational and widely-adopted method [45]. The AT process involves augmenting the training dataset with adversarial examples generated using known attack strategies. Specifically, it aims to solve a min-max optimization problem:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\|\delta\|_p \leq \epsilon} \mathcal{L}(f_\theta(x+\delta), y) \right], \tag{2}$$

Here, the inner maximization seeks the worst-case perturbation δ (bounded by ϵ under the ℓ_p-norm) that maximizes the loss $\mathcal{L}$, while the outer minimization updates the model parameters θ to minimize the expected loss over these adversarial examples.

Despite its effectiveness, standard AT has limitations: it is typically tailored to specific attack methods [20, 45] and consequently may exhibit weak performance against advanced, adaptive, or unseen adversarial attacks [11]. More advanced defense methods such as noise injection can be integrated with the AT framework, as we will introduce next.

2.2 Noise Injection as Defense

Previous studies have demonstrated that noise injection techniques can effectively enhance adversarial robustness [12, 25, 28, 44, 50, 62]. By injecting noise into the model, this approach acts as a form of regularization, increasing the difficulty for adversaries to generate effective perturbations [50]. However, existing noise injection methods [25, 28, 44, 62] share a common drawback: even being learnable with parameters, they consistently lead to a drop in clean accuracy, undermining the model's overall performance in benign scenarios. This issue remains unexplored and limits further advancements in noise injection techniques.

Moreover, current noise injection methods mostly focus on algorithm design while overlooking inference efficiency. This leads to additional computational overhead during inference, which results in higher energy consumption and latency [18]. These extra costs significantly hinder the deployment of noise-injection-based defenses, especially on resource-constrained edge computing devices.

2.3 Specialized NPU Architectures

Recent advances in specialized NPU architectures [26, 47, 63, 67] have significantly enhanced the efficiency of DNN inference by exploiting inherent sparsity in weights and activations, as well as low-precision arithmetic. These architectures effectively reduce redundant computations and data movements – enabling efficient inference on resource-constrained platforms.

However, the focus has primarily been on enhancing performance and energy efficiency, often neglecting adversarial robustness. Given the security challenge from adversarial attacks, it is imperative to reconsider NPU architectures to integrate robust defense mechanisms.

Thus, we aim for hardware-aware defense methods that can be integrated at the architecture level. The key question is how can we efficiently support defense methods. The scope of our work is on the relation of adversarial robustness and execution efficiency when using noise injection against adversarial attacks.

3 Motivation

This section presents our motivation by outlining three key perspectives. First, we introduce the Critical Pathway Hypothesis, where we observe that existing noise injection methods treat all neurons uniformly. This uniform application leads to a drop in clean accuracy. Then, we delineate the potential in efficiency gains via approximate computing for non-critical pathways. Finally, we pose the questions on how to re-purpose specialized NPU architectures to support critical pathway protection.

3.1 Critical Pathway Hypothesis

As discussed in Section 2.2, noise injection methods [12, 25, 28, 44, 50, 62] typically lead to a drop in clean accuracy. To understand this behavior, we draw inspiration from activation sparsity methods [35, 38, 57], emphasizing that not all activations contribute equally. Prior work [38, 41, 42] has consistently shown that important activation patterns vary across inputs, indicating that they are dynamic rather than static. These key activations form connected critical pathways for accurate predictions [31, 65]. Based on this observation, we further conduct an empirical study to compare the impact of noise injection on clean accuracy when applied to critical pathways (i.e., activations with large magnitude) versus non-critical pathways (i.e., activations with small magnitude).

We follow the settings from a prior study [12] that selects the ResNet-18 as the baseline model. To align with the adversarial attack scenario, the model is trained on CIFAR-10/100 datasets using overfitting adversarial training [52]. Next, we dynamically identify each model's critical and non-critical pathways through the *Top-K* selection method. Finally, we investigate these two pathways' sensitivity by gradually increasing the noise injection ratio, applying noise to the critical or non-critical pathways first, and evaluating their responses to noise injection.

As shown in Figure 2, we can observe the following insights: (1) *Critical pathways are more sensitive than non-critical pathways to the noise injected.* When injecting noise to critical pathways, clean accuracy begins to decrease at the very beginning, e.g., 20% noise injection causes about 3% clean accuracy drop. (2) *Non-critical pathways account for most of the activations.* When injecting noise

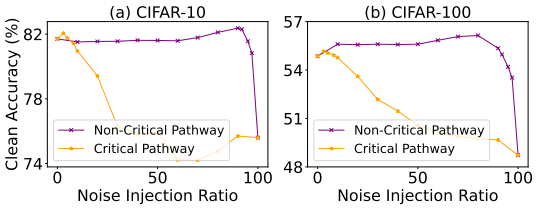

Figure 2: Impact of noise injection ratio on clean accuracy of ResNet-18 for CIFAR-10 and CIFAR-100. Injecting noise in non-critical pathways preserves accuracy better, while adding noise to critical pathways decreases accuracy.

to non-critical pathways, we can see that clean accuracy will not decrease significantly until the noise injection ratio reaches very high levels, e.g., 90%-93%.

Thus, we hypothesize that only a subset of activations that form pathways dynamically are essential for representation learning and highly sensitive to noise injection. Meanwhile, the remaining activations forming non-critical pathways can tolerate noise without compromising overall clean accuracy. From the critical pathway hypothesis, we aim for a noise injection method being selective to non-critical pathways while preserving critical pathways.

3.2 Opportunities with Approximation

Drawing from the insights in Section 3.1, an ideal noise injection method that preserves clean accuracy should (1) accurately identify critical pathways and (2) then efficiently inject noise only into the non-critical ones. A naive method would first execute the complete DNNs model layer-by-layer and then dynamically injects noise into the non-critical pathways, which can be done in a layer-wise manner. This naive approach, however, incurs redundant computations and data movements. Notably, the critical pathways account for only about 10% of the network on CIFAR-10/100, meaning that most of the operations on non-critical pathways are not necessary in the original format if these activations will be injected with noise. Moreover, generating noise demands additional hardware resources [18]. This raises a natural question: Can we bypass the precise computation of the full model while still injecting noise to the non-critical pathways?

This motivates us to explore the accurate and efficient detection method for protecting critical pathways and brings defensive noise to the non-critical ones. Rather than wasting more expensive computations for these non-critical pathways, we can approximate them with lower-cost computation [6]. Our approximation method stands out from prior methods such as Defensive Approximation [21]: On the one hand, ours can be integrated into the adversarial training framework and alleviated from the slow convergence as in Defensive Approximation. On the other hand, our method does not need a redesign of new hardware architectures. Instead, we seek to re-purpose the existing sparse architectures to support critical pathway protection, as we discuss next.

3.3 Re-purpose Specialized NPU Architectures

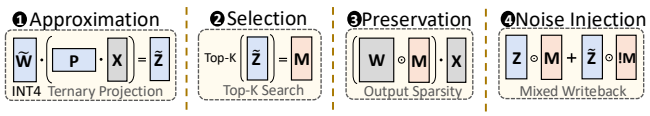

Figure 3: Operation mapping.

These challenges lead us to re-purpose specialized NPU architectures to improve execution efficiency when using noise injection as defense without intrusive hardware design. As described in Section 3.2, our algorithm has four key steps: Approximation, Selection, Preservation, and Noise Injection. Figure 3 illustrates how each step aligns directly with existing hardware capabilities (Step ❶ to Step ❹). This 1:1 correspondence ensures that our algorithm can be mapped onto typical NPU architectures without additional hardware modules. Specifically, we aim to address the following questions that can shape our architectural integration:

How to efficiently perform approximation at the hardware level? While dynamic activation sparsity accelerators effectively skip redundant neurons, efficiently injecting noise into non-critical pathways remains a challenge. Our architecture integrates Approximation (Step ❶) and Noise Injection(Step ❹) using lightweight and hardware-friendly designs. Inspired by mixed-precision and low-cost projection modules in prior work [1, 7, 12, 43], we use ternary random projection, enabling computation with only additions and subtractions rather than multiplications. In addition, since low-precision operations are commonly used, NPUs commonly include low-precision arithmetic to further improve efficiency. To realize noise injection, we construct the final output by mixing approximate results from the non-critical pathways with high-precision outputs from the critical pathways. However, this approach leads to writeback challenges due to the irregular memory locations of critical pathway activations. To address this, we explore three strategies: (1) Sequential Writeback: Write every value in memory address order; (2) Immediate Write to SRAM: Store all noise-injected activations, then overwrite with protected activations; (3) Sparse Storage: CSR-like, requiring hardware support.

How to avoid redundant computation and data movement? This integration directly corresponds to the Selection (Step ❷) phase, where non-critical pathways are excluded from computation, and the Preservation (Step ❸) phase, where critical pathways are retained for high-precision processing. For the Selection step, a top-K search can be used to identify critical pathways. Although this is inherently a search operation, it benefits from efficient vectorized implementations. To further improve efficiency, it's better to use threshold to achieve this. This approach avoid an extensive search through all data, allowing the thresholding operation to be vectorized efficiently and executed using reusable datapaths. For Preservation step, existing accelerators supporting dynamic activation sparsity [26, 47, 63, 67] can detect and skip near-zero activations at runtime. By integrating our critical pathways approach with such sparsity-supported architectures, we can selectively bypass redundant computations and data transfers for non-critical pathways without intrusive architecture redesign. This re-purposing of existing hardware significantly reduces overhead while retaining the performance gains of adversarial robustness.

Can we explore more hardware-friendly Pathways? The Critical Pathways mechanism can be further improved to handle the irregular distribution of critical activations and reduce dataflow complexities on hardware accelerators. In conventional design, the N:M sparsity pattern [49] offers a simpler route: it enforces regular blocks of zeros, leading to straightforward hardware implementations. However, integrating N:M sparsity with noise injection to further enhance efficiency and maintain adversarial robustness

remains unexplored. This gap motivates us to investigate more hardware-friendly pathways and their relationship to adversarial robustness. We will discuss the details in Section 6.

4 Pathway-Aware Selective Noise Injection

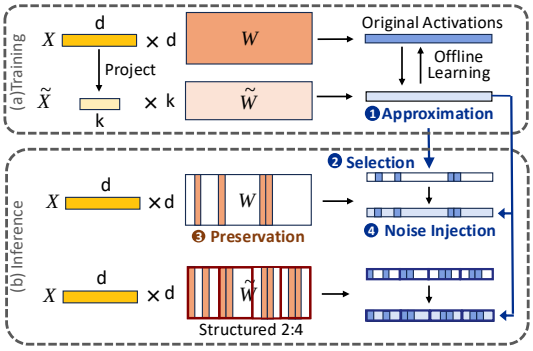

Figure 4: Algorithm Overview of SINAI.

As discussed in previous sections, an ideal method should accurately and efficiently identify critical pathways for protection while introducing defensive noise to non-critical ones. To achieve this, we introduce a learnable approximation module that precisely predicts the original pathways. This module serves as the foundation for identifying critical pathways, ensuring that computational resources are allocated efficiently. Instead of introducing additional hardware resources for noise injection, we leverage the intrinsic noise from the approximation module and replace non-critical pathways with its outputs.

Our overall algorithm is illustrated in Figure 4, in the following subsections, we first explain our offline learnable approximation module with the computation of approximations (Step ❶), followed by the critical pathways selection (Step ❷) and preservation (Step ❸) as well as efficient noise injection (Step ❹). We finally discuss how execution overhead is further minimized through hardware-aware structured pathways.

4.1 Learnable Approximation Module

As discussed in Section 3.2, the current approximation method [21] has two main drawbacks: it can not be integrated with adversarial training due to slow convergence, and it requires additional hardware resources to generate noise. We propose a new method that addresses these two challenges simultaneously. Specifically, our approach allows for smooth integration with adversarial training and incorporates noise in a hardware-friendly manner.

We formulate our layer-wise approximation using Random Projection [1, 7, 37], which can be efficiently implemented in hardware. In particular, we define $\tilde{Z} = \tilde{W}PX + \tilde{b}$, where $\tilde{W} \in \mathbb{R}^{n \times k}$ and $\tilde{b} \in \mathbb{R}^n$ are trainable parameters, X is the input, and P is a fixed ternary random projection matrix of size $k \times d$. During deployment, k is chosen to be $n/4$, allowing for a reduced computational footprint while preserving the same output dimension as Z. The matrix P is a ternary random projection matrix defined as $P \in \sqrt{\frac{3}{k}} \cdot \{-1,\ 0,\ 1\}^{k \times d}$, which introduces hardware-friendly noise without requiring additional generator.

To further improve efficiency, we incorporate quantization into this approximation module by reducing the bit-width of $\tilde{W}$ and $\tilde{b}$ to INT4 precision. While noise can be added by sampling from normal or uniform distributions, our approach directly injects noise through approximate values obtained from $\tilde{Z}$. This strategy not only simplifies the implementation but also substantially reduces computation.

When integrating our method in the adversarial scenario, we train our approximation module using a two-step process of Distillation followed by Fine-tuning. As shown in Figure 4 (a), during distillation, we freeze the original parameters and minimize the mean squared error (MSE) between the original layer output Z and its approximation $\tilde{Z}$:

$$\mathcal{L}_{MSE} = \frac{1}{B}||Z - \tilde{Z}||_2^2 = \frac{1}{B}||(WX + b) - (\tilde{W}PX + \tilde{b})||_2^2 \quad (3)$$

Here, B is the mini-batch size, and P is a fixed random projection matrix. For adversarial scenarios, we further fine-tune the approximation module by combining the original adversarial training (AT) loss with the MSE loss, the complete fine-tune procedure is detailed in Algorithm 1:

$$\mathcal{L}_{Final} = \mathcal{L}_{AT} + \alpha \mathcal{L}_{MSE}. \quad (4)$$

Here, α is a hyperparameter that balances robustness and approximation. This additional fine-tuning step ensures that our method more effectively learns critical pathways under adversarial attack. L_{AT} is the cross-entropy loss applied to adversarial samples, computed after inner-loop perturbation in adversarial training.

Algorithm 1 Pathway-Aware Adversarial Training

Require: Parameters W, b, $\tilde{W}$, $\tilde{b}$; projection matrix P; input X, label y; perturbation bound ϵ, step size η, attack steps T; threshold θ_{th}; weight α

Ensure: Final loss $\mathcal{L}_{\text{Final}}$

Generate pathway mask M using approximation
$X^{adv} \leftarrow X + \mathcal{U}(-\epsilon, \epsilon)$
for $t = 1$ to T **do**
  Compute Z with critical pathways:

$$Z_i \leftarrow \begin{cases} \phi(W[i,:]X^{adv} + b^i), & \text{if } M_i = 1 \\ \phi(\tilde{W}[i,:]PX^{adv} + \tilde{b}^i), & \text{otherwise} \end{cases}$$

  $g \leftarrow \nabla_X \mathcal{L}_{CE}(f(Z), y)$
  $X^{adv} \leftarrow \text{Clip}_{X,\epsilon}(X^{adv} + \eta \cdot \text{sign}(g))$
end for
Final forward pass with X^{adv}
$\tilde{Z} \leftarrow \tilde{W}PX^{adv} + \tilde{b}$
$Z_i \leftarrow M_i \cdot Z_i + !M_i \cdot \tilde{Z}_i$
$\mathcal{L}_{\text{AT}} \leftarrow \mathcal{L}_{CE}(f(Z), y)$
$\mathcal{L}_{\text{MSE}} \leftarrow \frac{1}{B}\left\|WX^{adv} + b - (\tilde{W}PX^{adv} + \tilde{b})\right\|_2^2$
$\mathcal{L}_{\text{Final}} \leftarrow \mathcal{L}_{\text{AT}} + \alpha \cdot \mathcal{L}_{\text{MSE}}$

4.2 Critical Pathways Preserved Noise Injection

After we obtain optimized approximation parameters (Step ❶), we can use the approximate values to identify critical pathways. More

importantly, we leverage them during inference to efficiently inject noise into each layer's activations.

For every layer, once we calculate approximated output $\tilde{Z}$, we can then use the values of $\tilde{Z}$ to evaluate whether activation in the individual layer is critical dynamically. In the case of irregular patterns of critical activations, an activation is regarded as critical if its approximation from $\tilde{Z}$ is above a certain percentile of all values. This identification process can be done by top-k or tuned thresholding on the validation dataset. Functionally, we represent the critical activations as a binary mask $M \in \{0,1\}^{j \times k}$, where 1 denotes a critical activation, which should be protected (Step ❷).

Once the critical activations are identified, layer-by-layer they form the critical pathways. We protect these critical pathways by triggering accurate computations (Step ❸) to maintain clean accuracy, while injecting noise via approximation into the remaining pathways (Step ❹). We formalize the outputs of a layer under our method as:

$$Z^{'}_{jk} = Z_{jk}M_{jk} + \tilde{Z}_{jk}(!M_{jk}) \tag{5}$$

, where $\tilde{Z}_{jk} = \tilde{W}_{jl}(P_{li}X_{ik}), Z_{jk} = W_{ji}X_{ik}$. Notably, protecting only a subset of activations at full precision shares similarity with sparse activation processing. Hence, specialized sparse architectures can be employed to reduce the overhead of protecting critical activations, making our approach both flexible and efficient.

4.3 Structured Noise Injection Pattern

While we can reduce execution overhead of protection of critical pathways, the unstructured patterns still increase the hardware design complexity and execution overheads from irregular data access and low data reuse, as discussed in Section 3.3. For an irregular noise injection at a given ratio, the distribution is not guaranteed to be in any form (uniform, clustered, patterned, etc.). This naturally demands significant overhead to handle the dynamic processing introduced, i.e., identifying and protecting critical pathways.

Inspired by sparsity-oriented designs, we hypothesize that noise injection granularity can be constrained in a manner similar to sparsity constraints.Specifically, we consider the $N : M$ fine-grained structured sparsity paradigm, wherein N elements are preserved within each $1 \times M$ block of a dense matrix. This approach supports more flexible patterns than conventional block sparsity. For instance, $1:2$ and $2:4$ structured techniques featured in NVIDIA Ampere GPUs [49] aim to deliver high efficiency and faster inference while maintaining accuracy. However, the impact of this techniques on robustness remains unexplored.

Motivated by this, we propose structured noise injection. As illustrated in Figure 4, this revised method selects N non-essential activations within each group of M activations to inject noise, enabling more efficient and hardware-friendly execution.

4.4 Theory Insights

We provide theoretical support for SINAI, which balances clean accuracy and robustness by using random projection to inject noise and select critical pathways, resulting in two noise sources.

The design of the predictor module is grounded in the Johnson-Lindenstrauss Lemma [29], which ensures that high-dimensional vectors can be projected into a lower-dimensional space while approximately preserving pairwise distances and inner products. This enables accurate inner product estimation in a compressed space with noise, allowing efficient critical pathways selection while maintaining clean accuracy.

After identifying the critical pathways, SINAI performs a masked combination at each layer by mixing the full-precision activations Z^l with low-cost approximations $\tilde{Z}^l$ reconstructed from the random projection predictor. Let $M^l \in \{0,1\}^{n^l}$ denote the binary mask that selects critical pathways (where 1 indicates a critical neuron). The injected noise at layer l is defined as $\eta^l = (1-M^l) \odot (\tilde{Z}^l - Z^l)$, where $\tilde{Z}^l$ is trained via MSE distillation to approximate Z^l. Under this training objective, the noise satisfies $\mathbb{E}[\eta^l \mid x, M^l] = 0$. Let the post-training approximation variance be defined as $\delta_l^2 := \max_j \mathrm{Var}[\tilde{Z}_j^l - Z_j^l]$. Assuming that each neuron is injected with noise independently with probability $\rho \in [0,1]$ (i.e., the *noise injection ratio*), the variance of the injected noise satisfies $\mathrm{Var}[\eta_i^l \mid x] = \rho \cdot \delta_l^2$. Thus, SINAI provides explicit, fine-grained control over the per-layer noise variance through the injection ratio ρ.

Theorem 1 (Gradient Perturbation Bound). *Consider a neural network f_θ that is L_0-Lipschitz continuous, and let its loss function gradient $\nabla_x L(\theta, x, y)$ be L_1-Lipschitz continuous. Let η^l denote the layer-wise noise injected by SINAI, and let the total injected noise variance across layers be: $\sigma^2 = \sum^l \rho \cdot \delta_l^2$. Then, the expected perturbation introduced by SINAI in the gradient satisfies the following bound:*

$$\mathbb{E}_\eta \left\| \nabla_x L_{\text{SINAI}}(\theta, x, y; \eta) - \nabla_x L_{\text{clean}}(\theta, x, y) \right\|_2 \leq L_0 \sigma.$$

Theorem 1 shows that the deviation between SINAI's perturbed gradient and the clean gradient is upperbounded by the aggregated noise standard deviation σ, which is directly governed by the noise injection ratio ρ. This result indicates that by increasing ρ, SINAI can intentionally amplify the discrepancy in the gradient used by adversarial optimizers, thereby degrading their effectiveness. This theoretical insight aligns with the principles behind Random Noise Defense [51], and further provides a formal justification for methods such as DNNShield [56], which introduce dynamic sparsity as a defense mechanism but lack explicit noise modeling. These demonstrate that SINAI provides a clean accuracy-friendly mechanism to balance robustness and accuracy through controllable noise injection guided by dynamic pathway selection.

5 Architectural Integration

In this section, we present the architectural design for integrating pathway-aware noise injection into a generic specialized neural processing unit (NPU) with sparsity and low-precision support. Our approach is grounded in hardware-friendly co-design principles, ensuring that the algorithm maps efficiently onto NPUs that exploit activation sparsity and reduced-precision arithmetic. As discussed in Section 3.3, to preserve generality and avoid overspecialization, the core components of our algorithm can be mapped to the dataflow and execution model of a generic NPU architecture. Based on this mapping, we further introduce architectural modifications that enable selective pathway-aware noise injection.

We begin by defining our baseline architecture, NPU-Base. This architecture consists of a core-based design, where each core features a vector unit that can perform matrix-vector multiplication,

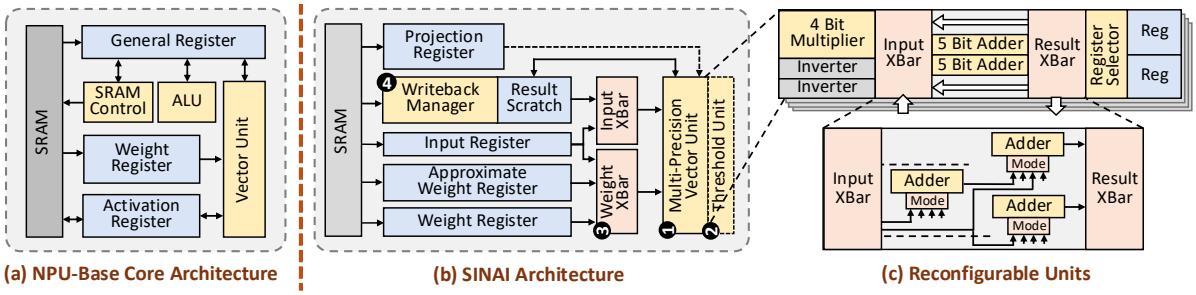

Figure 5: (a) Baseline NPU core architecture; (b) SINAI architecture; (c) Reconfigurable Unit with the support for multi-operation.

vector/general register files, and SRAM scratchpad. Figure 5(a) illustrates the architecture of an NPU-Base core. While the full NPU can scale with multiple cores, we normalize our analysis and performance comparisons on a per-core basis to isolate the benefits of our proposed extensions.

5.1 Reconfigurable Multi-Precision Design

Our SINAI algorithm employs three numerical formats – INT16, INT4, and ternary – which necessitate architectural support for multi-precision operations and computation skipping. Therefore, we consider another improved baseline, NPU-Specialized, which supports low-precision, projection, and activation sparsity.

A typical NPU-Specialized architecture might involve provisioning separate compute units for INT4 and ternary operations. Given their lightweight nature and small core footprint, this approach is viable. However, it duplicates resources instead of reusing hardware across similar operations. Upon decomposing the main arithmetic operations, we observe that all three modes can be expressed as variants of low-precision integer arithmetic. Specifically, INT4 MACs are composed of INT4 operators; INT16 MACs can be broken down into tiled INT4 operations; Ternary projection at INT4 resolution is signed additions/subtractions. For projection, though, we adopt INT5 as a safer intermediate precision less prone to overflowing.

As shown in Figure 5(b)❶, our SINAI architecture uses a multi-precision vector unit that co-locates the execution of INT16, INT4, and ternary operations. Motivated by previous observations, we construct the vector unit based on INT5 adder lanes. Shown in Figure 5(c), the multiple precisions are enabled by the adder fabric which provides flexible data routing between adders. As well, we use enhanced routing logic for INT16 multiplication to avoid explicit bit-shifting hardware in use in other multi-precision designs. This structure enables dynamic reconfiguration between different modes without incurring significant area overhead.

Note that our SINAI architecture is tightly integrated on top of NPU-Specialized, but with improved area and power efficiency. Table 2 list the overall core-level area and power comparisons.

5.2 Other Design Considerations

In addition to multi-precision support, several design considerations are required to fully integrate pathway-aware noise injection into the NPU-Specialized architecture.

As shown in Figure 5(b)❷, for the selection of critical pathways, we employ a threshold unit that operates in-line with the pipeline.

Table 2: Area and power estimations.

Item	Area (mm^2)	Power (mW)
NPU-Base	0.6457	162.6
NPU-Specialized	0.7448	163.1
SINAI	0.6833	169.3
SINAI-e	0.8250	249.1
Vector Unit	0.1007	–
- Arithmetic Units	0.0395	–
- Fabric Switching	0.0087	–
Registers	0.0205	–
Crossbars	0.0043	–

This threshold unit is directly coupled to the output of the vector unit, enabling runtime screening of activations as they are generated. By avoiding post-processing or global searches across activations, this approach ensures that the decisions are made with minimal latency and control overhead, and it naturally aligns with the low-precision, streaming nature of the specialized NPU.

Figure 5(b)❸ shows the basic Weight/Input Crossbars for the support of output activation sparsity. However, the dataflow introduces dynamic operations, which makes it difficult to maintain efficiency, particularly in weight reuse. As shown in Figure 6, the reuse of weights across columns can be unpredictable. Since reuse between adjacent columns is unpredictable, we choose to execute in the most reliable manner, with a column-major execution order. Although this reduces potential operation savings, it ensures consistent latency and predictable execution, similar to approximate computations. It involves a sequence of smaller $M \times V$ operations, constructing a concise weight matrix and fixing columns. This decision ensures consistent execution time.

Despite the irregularity of sparsity, the column is guaranteed to have a specific number of densely computed elements. We therefore use a column-major execution pattern which sparsely loads weight rows and calculate a single output column at a time. This ensures predictability. Then, to support the mixed-precision output pattern resulting from pathway-aware computation, we design a hybrid writeback scheme, with a writeback manager shown in Figure 5(b)❹. During the thresholding stage, two separate queues are generated: one for addresses containing high-precision (critical) activations and another for approximated (non-critical) ones. Critical outputs are written back in a sequential, blocking manner to preserve precision, while non-critical activations are written asynchronously to amortize memory bandwidth and reduce stall cycles. This dual-queue design avoids the complexity of dynamic lookahead mechanisms and maintains low control overhead, adding

no more than eight additional cycles beyond baseline multiplication latency under typical conditions.

6 Synergy on Structured Noise Injection

Although our algorithm can execute efficiently on the SINAI architecture, there is still room for performance improvement. A key limit exists from the unstructured selection of critical pathways, which introduces two major challenges for hardware execution: (1) limited data locality, leading to inefficiencies in weight reuse of output sparsity, and (2) irregular output patterns, which complicate and slowdown the writeback process.

To address these issues, we propose SINAI-e, an enhanced version of SINAI that incorporates structured noise injection as proposed in Section 4.3. This enforced structure directly compensates for the deficiencies of unstructured selections. As demonstrated in Figure 6, hardware execution is greatly simplified. In this section, we highlight the benefits of SINAI-e and detail the specific architectural enhancements. We focus on increasing throughput by increasing weight reuse, widening the vector unit, and simplifying the writeback process, rather than area or power savings.

6.1 Efficient Processing with Output Sparsity

Unlike the unstructured SINAI design, where weight reuse is sacrificed to enforce regularity, the structured design improves data locality and enables weight reuse across columns. This improves arithmetic intensity and allows for wider vector units. With structured design, each set of M weight vectors is reused for every output column, enabling concurrent processing of input columns. Additionally, the number of weights N is fixed, which ensures predictable fetch and usage rates for both inputs and weights. This helps reduce memory stalls caused by bandwidth limitations. As a result, SINAI-e achieves fewer memory accesses, higher compute throughput, and simpler control logic.

Hardware Modifications: To enable weight reuse, the weight-loading mechanism in SINAI needs to be replaced. Instead of constructing an abridged weight matrix, we must load all M (8) weights. With all 8 weights in the weight register, we then use input routing to supply the Vector Unit with the appropriate weights for each computed column. In this way, when progressing column to column, no weight loading is necessary.

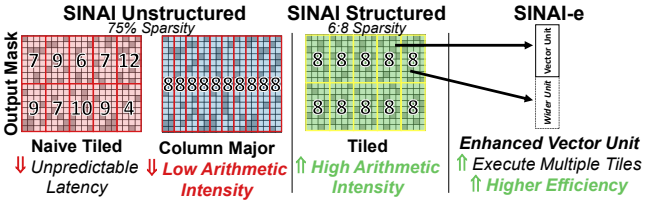

Figure 6: The impact of structure on execution pattern.

This locality enables us to choose an execution pattern that computes output matrices concurrently, rather than computing columns one at a time. As shown in Figure 6, this execution pattern enables us to widen the Vector Unit by a factor of 2 while guaranteeing saturation in the case of the most extreme noise injection, 7:8. This saturation is constrained by the joint input and weight bandwidth.

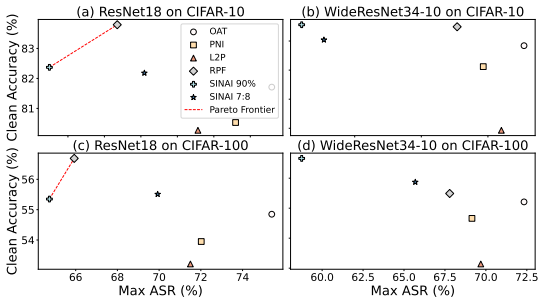

Figure 7: Comparison of clean accuracy and Max ASR across different defense methods on CIFAR-10/-100 using ResNet18 and WideResNet34-10. Each point represents a defense method, where lower Max ASR indicates better robustness.

6.2 Structured Writeback

In the SINAI design, the writeback process requires separate storage of critical and non-critical pathways. These are then combined for the final output matrix, resulting in a write-rewrite pattern that introduces unnecessary time and memory overhead. In contrast, structured noise injection can avoid this behavior altogether.

Due to the enforced regularity, structured writeback directly constructs the final output during the computation of critical pathways. As opposed to the potentially large gap between critical pathways, we have a guaranteed pattern in the window size M. Because the structured pattern selects N non-critical activations for every M outputs, we can reliably prefetch the non-critical pathways while computing the critical ones. As a result, control logic becomes significantly simpler compared to the complex coordination required in the SINAI unstructured.

Hardware Modifications: With the timing and location of non-critical pathways fully predictable, the need for complex address translation logic is removed. Memory addresses can now be computed directly at execution time. Furthermore, because the end of each critical pathway's execution is known in advance, the writeback unit only requires a small scratch memory to temporarily manage the final mixed outputs. This leads to lower hardware complexity and reduced memory footprint during execution.

7 Evaluation

Our evaluation shows that: (1) SINAI enhances robust accuracy compared with baseline models using adversarial training; (2) With the architectural integration for SINAI, we improve execution efficiency in terms of performance speedup and energy saving; (3) Further equipped with architectural support for structured patterns of selective noise injection, we obtain additional efficiency gains.

7.1 Evaluation on Adversarial Robustness

We conduct experiments against different adversarial attacks to evaluate our algorithm's ability to dynamically preserve critical pathways. The results validate our algorithm's robustness as well as support our hypothesis. Measuring the *response* under adversarial attacks gives us the insight into SINAI's robustness.

7.1.1 Algorithm Evaluation Methodology. In our experiments, we examine the performance of ResNet18 [24] and WideResNet34-10 [66] on the CIFAR-10 and CIFAR-100 datasets [33], as well as

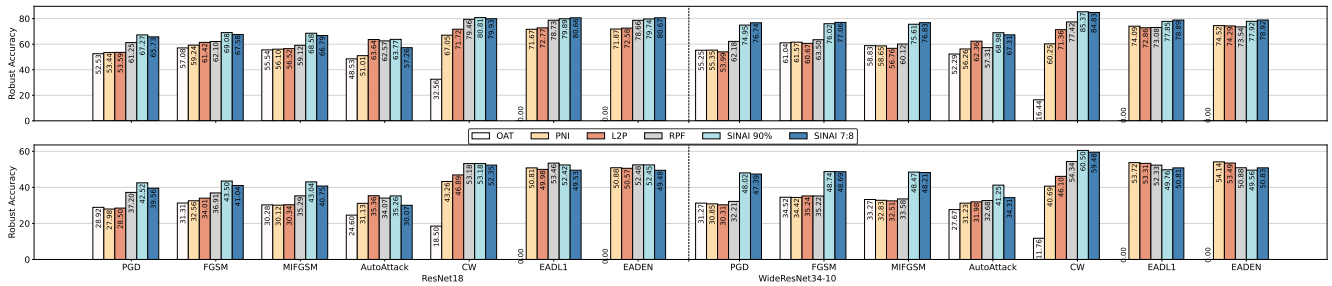

Figure 8: Comparison of the SINAI algorithm's robustness on CIFAR-10 (top) and CIFAR-100 (bottom) using ResNet18 and WideResNet34-10, evaluated under various adversarial attack strategies.

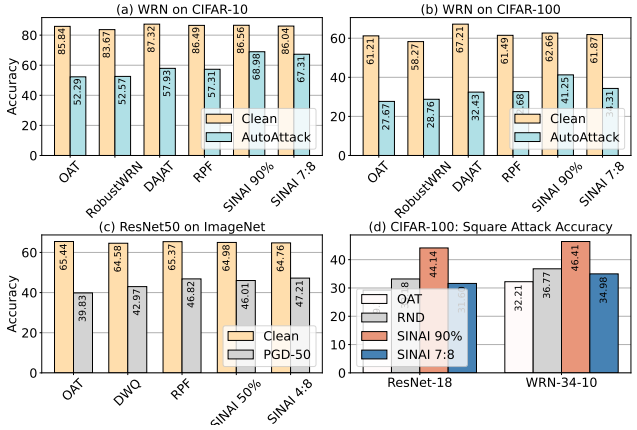

Figure 9: (a) (b) Comparison with other defense methods of WRN on CIFAR-10/100. (c) Robust performance of ResNet-50 on ImageNet. (d) Results under square attack for ResNet-18 and WideResNet34-10 on CIFAR-100.

ResNet50 on ImageNet and TinyImageNet [55]. To evaluate adversarial robustness, we implement various attack methods, including Projected Gradient Descent (PGD) [45], Fast Gradient Sign Method (FGSM) [20], Momentum-based Iterative Fast Gradient Sign Method (MIFGSM) [13], AutoAttack [11], CWL2[8] and EAD[9]. We assess robustness using two metrics: Robust Accuracy and the Max Attack Success Rate (Max ASR), Max ASR $= \max_i \left(\frac{|\{x_{\text{adv},i} : f(x_{\text{adv},i}) \neq y\}|}{|\{x_{\text{adv},i}\}|} \right)$ which represents the highest success rate among all attack methods. We utilize the torchattacks [32] library in Pytorch to deploy these attack methods, adhering to the parameters specified in the library's protocol. Specifically, for FGSM, PGD, MIFGSM, and AutoAttack, we set the maximum perturbation size ϵ to 8/255 and use a step size of 2/255 for PGD and MIFGSM. PGD is conducted over 20 steps, while MIFGSM uses 5 steps. For AutoAttack, we use the default setting of 100 attack steps. We follow the default configurations from the original CW and EAD implementations, with $\kappa = 0$, lr = 0.01, binary search steps is 9, 100 max iterations and $\beta = 0.001$. On ImageNet and TinyImageNet, ϵ is set to 4/255, with 10 and 50 steps for the PGD attack. For the comparison of adversarial robustness, we use overfitting adversarial training (OAT) method [52] as a baseline adversarial training framework. Our algorithm part primarily involves noise injection, so we compare various noise injection methods, including Parametric Noise Injection (PNI) [25], and Learn2Perturb (L2P) [28]. Additionally, we include a recent method, Random Projection Filters (RPF) [12], which is also a noise injection technique. We reproduce all the aforementioned methods within the OAT framework for fair comparison.

Table 3: Adaptive PGD attack on ResNet-18 under different masking and loss strategies.

Method	CIFAR-10			CIFAR-100		
	Clean	A1	A2	Clean	A1	A2
OAT	81.71	52.53	52.53	54.85	28.92	28.92
SINAI 90%	82.37	56.21	60.19	55.35	31.73	35.04
SINAI 7:8	82.18	53.93	59.21	55.51	29.06	32.11

7.1.2 Robustness Results. For our main results, we progressively increase the noise ratio until a drop in clean accuracy is observed. On CIFAR-10 and CIFAR-100, our method applied to both ResNet18 and WideResNet34-10 can tolerate up to a 90% or 7:8 noise ratio without any degradation in clean accuracy. Figure 7 shows the Pareto trend between clean accuracy and Max ASR. Injecting noise into non-critical pathways significantly enhances robustness while preserving clean accuracy, outperforming other uniform noise injection methods. Detailed results for each attack method are provided in Figure 8, where our method consistently achieves superior robust accuracy and maintains comparable or better clean accuracy across all networks and datasets.

Results on ResNet18. Specifically, for ResNet18, our SINAI 90% setting maintains a clean accuracy of 82.37% and significantly enhances robust accuracy over OAT. The improvements are 14.74% for PGD^{20}, 12.00% for FGSM, 13.04% for MIFGSM, and 15.24% for AutoAttack. Compared to noise injection-based methods, our approach shows a 6.02% to 9.46% improvement under PGD^{20}, FGSM, and MIFGSM, and achieves comparable robust accuracy against AutoAttack. Notably, the structured 7:8 noise injection ratio achieves performance on par with the 90% setting, showing its potential for robustness with more efficiency gain. CW and EAD attacks rely on precise optimization objectives, making them ineffective or unstable against noise injection–based defenses. Results on CIFAR-100 follow similar trends, further demonstrating the effectiveness of our method on more complex tasks

Results on WideResNet34-10. For wider models, our method achieves even better improvements. When using WideResNet34-10 on CIFAR-10, SINAI improves robust accuracy by 12.77% under

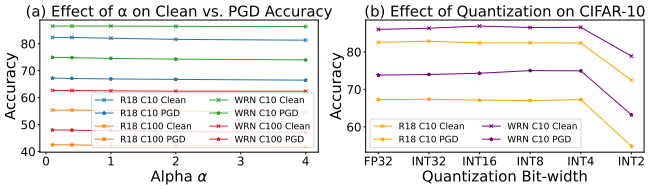

Figure 10: (a) Ablation study of α (b) Ablation study of Quantization Bit-width.

PGD^{20} and 11.67% under AutoAttack compared to RPF. On CIFAR-100, our method similarly demonstrates strong performance, achieving improvements of 15.81% under PGD^{20} and 8.57% under AutoAttack over RPF. This highlights SINAI's ability to scale effectively across different model sizes and maintain strong performance.

Overall, these results demonstrate the efficacy of SINAI in improving adversarial robustness while preserving clean accuracy. In all cases, SINAI leverages the advantages of noise injection and improves robustness. Meanwhile, it maintains all of the original model's accuracy on clean data, suggesting that critical pathways are minimally disrupted and pass the important information.

7.1.3 Comparison with other Defense Methods. To further evaluate the adversarial robustness of our method, we compare SINAI against several recent state-of-the-art defenses that do not rely on noise injection, including RobustWRN [27] and DAJAT [2]. We show these results in Figure 9 (a) and (b) using the WideResNet34-10 on CIFAR-10 and CIFAR-100 datasets. We conduct all experiments using the comprehensive AutoAttack and reproduce all baseline methods for a fair comparison. We also compare with noise injection based method RND [51] under black-box attack in Figure 9 (d).

Compared with these prior defense methods, SINAI 90% outperforms other baseline methods on WideResNet34-10, with improvements ranging from 11.05% to 16.69% on CIFAR-10 and 8.57% to 13.58% on CIFAR-100 under AutoAttack. In the structured noise injection setting, SINAI 7:8 demonstrates gains across both datasets, highlighting the effectiveness of our method. Also, SINAI achieves higher robustness than RND across multiple architectures.

7.1.4 Larger Model and Dataset. To further evaluate our method's generability and scalability, we also apply our method to ResNet50 on the ImageNet dataset. This setting allows us to explore its effectiveness on larger neural networks and more complex input with higher resolution. For baseline comparison, we reproduce OAT, RPF, and Double-Win Quantization [15]. In this setting, SINAI supports a 50% random noise injection ratio and a structured 4:8 pattern, both without noticeable loss in clean accuracy. As shown in Figure 9 (c), our method achieves improvements of 6.18% to 7.38% in robust accuracy under the PGD-50 attack compared to the OAT baseline, outperforming all other methods. These results indicate our SINAI method's potential for real-world applications, where both model size and input complexity present significant robustness challenges.

7.1.5 Under Adaptive Attack. To evaluate the robustness of SINAI against stronger, defense-aware attack methods, we design two adaptive attacks [59], each targeting one of SINAI's two noise sources: approximation noise and dynamic pathway selection.

Table 4: Robust accuracy under PGD and AutoAttack with EOT (M = 10) on ResNet-18.

Attack	CIFAR-10			CIFAR-100		
	OAT	SINAI 90%	SINAI 7:8	OAT	SINAI 90%	SINAI 7:8
PGD	52.53	67.27	65.73	28.92	42.52	39.56
EOT+PGD (M=10)	53.27	67.78	66.02	29.11	42.69	39.79
AutoAttack	48.53	63.77	57.26	24.60	35.26	30.07
EOT+AutoAttack (M=10)	48.64	64.18	57.98	24.69	36.09	31.01

Table 5: Robustness evaluation of ViT-S-16 on CIFAR-100 under various adversarial attacks.

Method	Clean	FGSM	PGD	MIFGSM	AutoAttack
Adversarial Training	58.18	30.76	28.32	29.81	23.18
ReiT [19]	60.26	31.37	29.54	30.78	25.91
Attention 90%	62.85	33.98	30.64	32.64	25.84
Attention 7:8	62.30	33.67	30.27	32.17	25.55
FFN 90%	56.37	40.39	37.22	39.03	29.71
FFN 7:8	57.22	36.91	34.19	35.31	27.62
Both 90%	55.45	39.92	36.68	38.26	30.38
Both 7:8	55.48	36.70	33.88	35.29	28.20

A1: We assume attackers are aware of the approximation noise injected by random projection. We modify the PGD objective to maximize the difference between clean and noisy activations:

$$\min_{\delta} \left(\mathcal{L}_C(x+\delta, y_t) - \mathcal{L}(A(Noisy(x+\delta)), A(x+\delta))\right)$$

where $\mathcal{L}_C$ is the classifier's loss, $Noisy$ is the function that applies noise to the neurons, A is the activation.

A2: The attacker is assumed to know the exact critical pathway mask and avoids gradients from non-critical pathways. During PGD, the gradients of non-critical pathways are zeroed out.

Both attacks are implemented under the same PGD-20 ($\epsilon = 8/255$ and step size = 2/255.) We can observe from Table 3 that (1) SINAI exhibits strong robustness even under adaptive attack. (2) Noise introduced by random projection contributes more to robustness than that from dynamic pathway selection.

7.1.6 Ablation Study. We also provide ablation studies corresponding to the design of the loss weight α and the precision of prediction module, as shown in Figure 10. We can observe that varying α has minimal impact on model's performance, with only slight drops, as our method requires only a few fine-tuning steps. The prediction module maintains stable performance down to INT4, confirming that its quantization does not affect robustness.

7.1.7 EOT Attack. To explore whether randomness in SINAI can be overcome by Expectation over Transformation (EOT), we follow the maximum EOT iteration setting in RND [51] and evaluate under EOT enhanced PGD and AutoAttack with $M = 10$ forward passes. These results suggest that the dual source noise in SINAI expands the gradient estimation space, making it harder for EOT-based attacks to converge.

7.1.8 Discussion on Vision Transformer. We conduct a study that applies of our algorithm to Vision Transformer. Specifically, we choose the ViT-S-16 [14] model on CIFAR-100 and apply our method to three different modules: the Q and K matrices of the Attention block, the first layer of the FFN block, and both combined. For each configuration, we test two sparsity levels: 90% (irregular) and

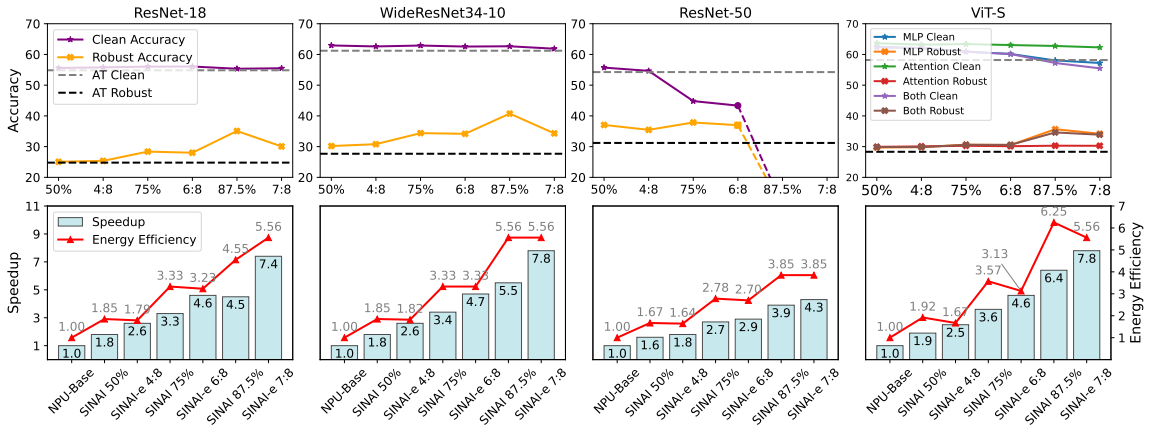

Figure 11: Clean and robust accuracy (top), and speedup and energy efficiency (bottom), across different noise injection ratios.

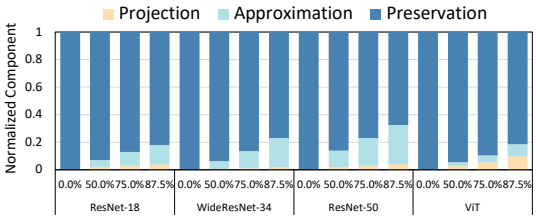

Figure 12: The energy breakdown of each stage by noise ratio & model. Both the projection and approximate stage have relatively low contributions to overall energy. ResNet50 uses smaller kernels, which limits the effectiveness of SINAI's compute pattern.

7:8 (structured). We use the same attack settings as in previous experiments, and adopt ReiT [19], which introduces noise at the token level in the Attention block, as our baseline.

As listed in Table 5, our exploration reveals several key insights: First, in terms of clean accuracy, the Attention-only injection shows the highest clean accuracy, with 62.85% (90%) and 62.30% (7:8), even slightly surpassing the ReiT baseline (60.26%). This indicates that attention in ViT are more resilient to noise, and may even benefit from the regularization effect of SINAI. Second, injecting noise into the FFN block results in stronger adversarial robustness than that in Attention block alone. For example, under AutoAttack, FFN 90% achieves 29.71% accuracy compared to 25.84% for Attention 90%. This suggests that the FFN layers are more influential targets for adversarial robustness in ViT models. Lastly, structured noise (1:8) performs competitively with irregular noise, often with only a small drop in robustness.

7.1.9 Noise Injection Ratio vs. Accuracy. We further explore how clean and robust accuracy change with different noise injection ratios, as shown in Figure 11 (top), across ResNet-18, WideResNet34-10, and ViT-S on CIFAR-100, and ResNet-50 on TinyImageNet. We evaluate three ratios: 50% (4:8), 75% (6:8), and 87.5% (7:8). For models on the CIFAR-100 dataset, we observe that clean accuracy remains relatively stable across all injection ratios. At the same time, robust accuracy improves steadily as more noise is injected, showing the effectiveness of SINAI in enhancing robustness with minimal accuracy loss. On the TinyImageNet dataset, clean accuracy starts high but drops significantly at higher injection ratios. This suggests that deeper models on more complex datasets are more sensitive to aggressive approximation. We attribute this to the varying sensitivity of individual layers, which may require layer-wise tuning of the noise injection ratio for optimal performance.

7.2 Evaluation of Execution Efficiency

In this section, we evaluate how integrating SINAI into the hardware design improves execution efficiency. With the support of the optimizations described before, our method achieves higher speedup performance and energy efficiency.

7.2.1 Hardware Experiment Methodology. We construct a cycle-accurate simulator from the ground up, which faithfully represents each element of execution for every dataflow. This simulator is composed of a set of methods, each possessing dedicated counters to track specific statistics and update upon each execution. By abstracting these methods, we devise instructions for the execution controller. We then use this simulator to calculate the various statistics for each instruction and abstract them into our pseudo-compiler. In this way, we are able to synthesize the various performance metrics based on model parameters.

Our power and area analyses are from multiple sources. We use CACTI for SRAM modeling. For other components, we implement our design in SystemVerilog. We then synthesize our design via Synopsys Design Compiler using the FreePDK 45nm PDK. We estimate the output capacitance of each component, considering both the context in which it was used and the input capacitance of other library-based logic circuits.

We evaluate the hardware-level efficiency of SINAI across three deep neural networks: ResNet18, WideResNet34-10, and ResNet50. For ViT-S-16, due to its shorter input sequence length, we only apply our evaluation to the FFN layers. For each model, we use injection ratios same as before. We analyze the impact of these configurations on three key metrics: speedup, energy, and energy-delay product (EDP). These metrics are used to demonstrate the effectiveness and efficiency of SINAI.

7.2.2 Performance Speedup. To better understand the performance implications of SINAI's design space, we compare normalized execution time across different noise-injection ratios mentioned before, as illustrated in the bar plots in Figure 11 (bottom). These results

not only validate the architectural efficiency of SINAI but also shed light on how noise patterns influence runtime behavior.

For irregular pattern, SINAI-base achieves *1.6× to 1.9×* speedup under 50% noise-injection ratio across the models. As the injection ratio increases, the benefits become more pronounced: Speedups under high ratios of 75% and 87.5% reachs *2.7× to 3.6×* and *3.9× to 6.4×*, respectively.

SINAI-e delivers improved speedup through structured design. Increased speedups resulted from the wider & more utilized VPU. However, not all models have the same performance gains. For ResNet50, layer dimensions do not map well to the altered dataflow, meaning that there was marginal improvement over SINAI-base.

7.2.3 Energy Efficiency Analysis. SINAI enhances energy efficiency by reducing computations on non-critical pathways, as shown in the curve plot in Figure 11 (bottom). For instance, at a 50% noise-injection level, SINAI reduces total energy consumption to nearly 60% of the baseline. Higher injection ratios lead to even greater savings, as fewer pathways require energy-intensive computation.

With the structured hardware SINAI-e, it is not surprising to see that the energy consumption is around equivalent, with slight increases observed in some layers. Although structured injection leads to more speedup by using hardware resources more efficiently, the extra power used by the wider Vector Unit and additional control logic balances out these gains. As a result, the total energy consumption remains nearly unchanged.

To further characterize why performance scales so strongly, we analyze the energy breakdown across execution stages in Figure 12. In this analysis, we divide the main operations of the SINAI algorithm into three stages: projection matrix multiplication, approximate module computation, and preservation of critical pathways. Due to the aggressive nature of the approximation scheme, the energy consumption of these stages is minimal in many cases. With this, it is clear that further energy gains should result from reducing the cost of the preservation stage.

7.2.4 Analysis of the Structured Gains. SINAI-e benefits significantly from structured noise-injection patterns. Through our analysis, we find three key benefits lead to the gains. Firstly, guaranteed values across row blocks reduce SRAM usage from increased weight reuse. Secondly, more weight reuse mitigates the extra SRAM accesses for partial results. Lastly, with enhanced locality, execution can be parallelized further, which enables a wider Vector Unit which does not require higher SRAM bandwidth.

7.2.5 Comparison with Other Methods on EDP. We compare multiple robust methods for accuracy and efficiency in Figure 13. First, there are several methods which provide increased robustness, but do not affect the underlying execution. These remain fixed at an energy-delay product (EDP) reduction of 1, which is neither better nor worse. 2-in-1 [16] randomly varies execution precision to disrupt attacks. However, for efficiency, it has limited upside due to the propensity of higher precisions to dominate execution power and time. Finally, SINAI and SINAI-e show high accuracy and high EDP reduction due to the joint optimization of both. This highlights how we can achieve both robust accuracy and execution efficiency.

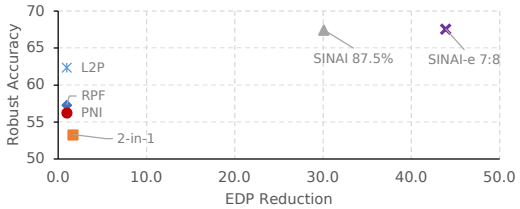

Figure 13: Robust accuracy and efficiency across methods. SINAI achieves the highest robust accuracy and energy-delay product reduction.

8 Related Work

Noise Injection Defense: Noise injection is an effective defense strategy where noise is introduced into DNNs during training or inference as a regularization term [25, 28, 44, 50, 62, 64]. For instance, PNI [25] applies learnable Gaussian noise to weights, inputs, and activations to explore noise injection granularity. Building on PNI, Learn2Perturb [28] progressively optimizes noise distributions for better robustness. However, these methods use uniform noise injection, which often harms clean accuracy. Random Projection Filters (RPF) [12] uses random projection to inject noise into a subset of filters, but incurs computational overhead, the need to train from scratch, and imprecise filter selection. In contrast, our method can identify and inject noise to only non-critical pathways. Beyond algorithm-level defenses, hardware-level noise has also been explored [10], but offers limited robustness improvement and lacks fine-grained control.

Sparsity-Based Defense: Prior work connects weight sparsity to improved robustness [30] via reduced Lipschitz constants [23], or leverages adversarial-aware pruning [39]. However, these approaches rely on static, input-agnostic pruning and offer limited robustness gains. DNNShield [56] introduces dynamic sparsity by randomly resampling weights per inference, but lacks input awareness and may disrupt critical paths, hurting clean accuracy.

Quantization and Robustness: Quantization has been widely explored for efficiency, as in OLIVE [22] and DRQ [58], but most works focus on the weight or activation distribution and do not consider robustness. DQ [40] and GRQR [3] improve robustness by controlling gradient magnitude through defensive quantization. 2-in-1 Accelerator [15] introduces randomness via mixed-precision noise injection, aiming to simulate defense behavior. However, these methods rely on static precision settings or require nontrivial hardware changes, and their robustness gains are limited.

Robust DNNs Accelerators: Addressing adversarial attacks, robust accelerators leverage hardware designs to go beyond purely algorithmic detection [17, 60, 61]. More related is on enhance robustness, e.g., 2-in-1 [16] suggests a random precision switch per layer, though still uniformly applied, possibly affecting essential activations. Others, like DeepFense [53] and DNNGuard [60], detect attacks by embedding unique layers or combining a CPU with a specialized DNN accelerator. However, such designs can increase area usage, potentially adding latency and diminishing efficiency.

9 Conclusion

In this paper, we introduce SINAI, an algorithm-architecture co-design framework that advances the state of the art in adversarial

robustness through hardware-efficient selective noise injection. By identifying and preserving critical pathways online and only injecting noise into non-critical pathways, SINAI enhances robustness without compromising clean accuracy. We demonstrate that SINAI integrates naturally with generic NPU architectures, with lightweight architectural modifications – such as multi-precision adder fabric and threshold-based selection unit. We further explore structured noise injection as a synergistic co-design that achieves more efficiency gain at on-par robustness. We expect *SINAI* to bridge the gap between robust algorithm and resource-constrained deployments – highlighting the potential of architectural integration of defense methods. Future work could further investigate hardware-aware adversarial defenses that leverage existing architectural features, without intrusive hardware modifications.

References

[1] Dimitris Achlioptas. 2001. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 274–281.

[2] Sravanti Addepalli, Samyak Jain, et al. 2022. Efficient and effective augmentation strategy for adversarial training. *Advances in Neural Information Processing Systems* 35 (2022), 1488–1501.

[3] Milad Alizadeh, Arash Behboodi, Mart Van Baalen, Christos Louizos, Tijmen Blankevoort, and Max Welling. 2020. Gradient eII_1 regularization for quantization robustness. *arXiv preprint arXiv:2002.07520* (2020).

[4] Abdulmalik Alwarafy, Khaled A Al-Thelaya, Mohamed Abdallah, Jens Schneider, and Mounir Hamdi. 2020. A survey on security and privacy issues in edge-computing-assisted internet of things. *IEEE Internet of Things Journal* 8, 6 (2020), 4004–4022.

[5] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. 2020. Square attack: a query-efficient black-box adversarial attack via random search. In *European conference on computer vision*. Springer, 484–501.

[6] Filipe Betzel, Karen Khatamifard, Harini Suresh, David J Lilja, John Sartori, and Ulya Karpuzcu. 2018. Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 1–32.

[7] Ella Bingham and Heikki Mannila. 2001. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 245–250.

[8] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*. Ieee, 39–57.

[9] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. 2018. Ead: elastic-net attacks to deep neural networks via adversarial examples. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[10] Sai Kiran Cherupally, Adnan Siraj Rakin, Shihui Yin, Mingoo Seok, Deliang Fan, and Jae-sun Seo. 2021. Leveraging noise and aggressive quantization of in-memory computing for robust dnn hardware against adversarial input and weight attacks. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 559–564.

[11] Francesco Croce and Matthias Hein. 2020. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International conference on machine learning*. PMLR, 2206–2216.

[12] Minjing Dong and Chang Xu. 2023. Adversarial Robustness via Random Projection Filters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4077–4086.

[13] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. 2018. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 9185–9193.

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*. https://openreview.net/forum?id=YicbFdNTTy

[15] Yonggan Fu, Qixuan Yu, Meng Li, Vikas Chandra, and Yingyan Lin. 2021. Double-win quant: Aggressively winning robustness of quantized deep neural networks via random precision training and inference. In *International Conference on Machine Learning*. PMLR, 3492–3504.

[16] Yonggan Fu, Yang Zhao, Qixuan Yu, Chaojian Li, and Yingyan Lin. 2021. 2-in-1 accelerator: Enabling random precision switch for winning both adversarial robustness and efficiency. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 225–237.

[17] Yiming Gan, Yuxian Qiu, Jingwen Leng, Minyi Guo, and Yuhao Zhu. 2020. Ptolemy: Architecture support for robust deep learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 241–255.

[18] Karthik Ganesan, Viktor Karyofyllis, Julianne Attai, Ahmed Hamoda, and Natalie Enright Jerger. 2023. Dinar: Enabling distribution agnostic noise injection in machine learning hardware. In *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*. 38–46.

[19] Huihui Gong, Minjing Dong, Siqi Ma, Seyit Camtepe, Surya Nepal, and Chang Xu. 2024. Random entangled tokens for adversarially robust vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 24554–24563.

[20] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[21] Amira Guesmi, Ihsen Alouani, Khaled N Khasawneh, Mouna Baklouti, Tarek Frikha, Mohamed Abid, and Nael Abu-Ghazaleh. 2021. Defensive approximation: securing cnns using approximate computing. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 990–1003.

[22] Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2023. Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–15.

[23] Yiwen Guo, Chao Zhang, Changshui Zhang, and Yurong Chen. 2018. Sparse dnns with improved adversarial robustness. *Advances in neural information processing systems* 31 (2018).

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

[25] Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. 2019. Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 588–597.

[26] Kartik Hegde, Hadi Asghari-Moghaddam, Michael Pellauer, Neal Crago, Aamer Jaleel, Edgar Solomonik, Joel Emer, and Christopher W Fletcher. 2019. Extensor: An accelerator for sparse tensor algebra. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 319–333.

[27] Hanxun Huang, Yisen Wang, Sarah Erfani, Quanquan Gu, James Bailey, and Xingjun Ma. 2021. Exploring architectural ingredients of adversarially robust deep neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 5545–5559.

[28] Ahmadreza Jeddi, Mohammad Javad Shafiee, Michelle Karg, Christian Scharfenberger, and Alexander Wong. 2020. Learn2perturb: an end-to-end feature perturbation learning to improve adversarial robustness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1241–1250.

[29] William B Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics* 26, 189-206 (1984), 1.

[30] Artur Jordao and Hélio Pedrini. 2021. On the effect of pruning on adversarial robustness. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1–11.

[31] Ashkan Khakzar, Soroosh Baselizadeh, Saurabh Khanduja, Christian Rupprecht, Seong Tae Kim, and Nassir Navab. 2021. Neural response interpretation through the lens of critical pathways. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 13528–13538.

[32] Hoki Kim. 2020. Torchattacks: A pytorch repository for adversarial attacks. *arXiv preprint arXiv:2010.01950* (2020).

[33] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

[34] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world.

[35] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. 2020. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In *International Conference on Machine Learning*. PMLR, 5533–5543.

[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[37] Ping Li, Trevor J Hastie, and Kenneth W Church. 2006. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 287–296.

[38] Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, et al. 2022. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. *arXiv preprint arXiv:2210.06313* (2022).

[39] Ningyi Liao, Shufan Wang, Liyao Xiang, Nanyang Ye, Shuo Shao, and Pengzhi Chu. 2022. Achieving adversarial robustness via sparsity. *Machine Learning* (2022), 1–27.

[40] Ji Lin, Chuang Gan, and Song Han. 2019. Defensive quantization: When efficiency meets robustness. *arXiv preprint arXiv:1904.08444* (2019).

[41] Junjie Liu, Zhe Xu, Runbin Shi, Ray CC Cheung, and Hayden KH So. 2020. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. *arXiv preprint arXiv:2005.06870* (2020).

[42] Liu Liu, Zheng Qu, Zhaodong Chen, Yufei Ding, and Yuan Xie. 2021. Transformer acceleration with dynamic sparse attention. *arXiv preprint arXiv:2110.11299* (2021).

[43] Liu Liu, Zheng Qu, Lei Deng, Fengbin Tu, Shuangchen Li, Xing Hu, Zhenyu Gu, Yufei Ding, and Yuan Xie. 2020. DUET: Boosting deep neural network efficiency on dual-module architecture. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 738–750.

[44] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. 2018. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 369–385.

[45] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

[46] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*.

[47] Nandeeka Nayak, Toluwanimi O Odemuyiwa, Shubham Ugare, Christopher Fletcher, Michael Pellauer, and Joel Emer. 2023. TeAAL: A Declarative Framework for Modeling Sparse Tensor Accelerators. *arXiv preprint arXiv:2304.07931* (2023).

[48] Wei Niu, Xiaolong Ma, et al. 2020. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 907–922.

[49] Nvidia. 2020. NVIDIA A100 Tensor Core GPU Architecture. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

[50] Rafael Pinot, Laurent Meunier, Alexandre Araujo, Hisashi Kashima, Florian Yger, Cédric Gouy-Pailler, and Jamal Atif. 2019. Theoretical evidence for adversarial robustness through randomization. *Advances in neural information processing systems* 32 (2019).

[51] Zeyu Qin, Yanbo Fan, Hongyuan Zha, and Baoyuan Wu. 2021. Random noise defense against query-based black-box attacks. *Advances in Neural Information Processing Systems* 34 (2021), 7650–7663.

[52] Leslie Rice, Eric Wong, and Zico Kolter. 2020. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning*. PMLR, 8093–8104.

[53] Bita Darvish Rouhani, Mohammad Samragh, Mojan Javaheripi, Tara Javidi, and Farinaz Koushanfar. 2018. Deepfense: Online accelerated defense against adversarial deep learning. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[54] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics* 9 (2021), 53–68.

[55] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Fei-Fei Li. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115 (2015), 211–252.

[56] Mohammad Hossein Samavatian, Saikat Majumdar, Kristin Barber, and Radu Teodorescu. 2022. DNNShield: Dynamic Randomized Model Sparsification, A Defense Against Adversarial Machine Learning. *arXiv preprint arXiv:2208.00498* (2022).

[57] Yixin Song, Haotong Xie, Zhengyan Zhang, Bo Wen, Li Ma, Zeyu Mi, and Haibo Chen. 2024. Turbo sparse: Achieving llm sota performance with minimal activated parameters. *arXiv preprint arXiv:2406.05955* (2024).

[58] Zhuoran Song, Bangqi Fu, Feiyang Wu, Zhaoming Jiang, Li Jiang, Naifeng Jing, and Xiaoyao Liang. 2020. Drq: dynamic region-based quantization for deep neural network acceleration. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1010–1021.

[59] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. 2020. On adaptive attacks to adversarial example defenses. *Advances in neural information processing systems* 33 (2020), 1633–1645.

[60] Xingbin Wang, Rui Hou, Boyan Zhao, Fengkai Yuan, Jun Zhang, Dan Meng, and Xuehai Qian. 2020. Dnnguard: An elastic heterogeneous dnn accelerator architecture against adversarial attacks. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 19–34.

[61] Xingbin Wang, Boyan Zhao, Rui Hou, Amro Awad, Zhihong Tian, and Dan Meng. 2021. NASGuard: a novel accelerator architecture for robust neural architecture search (NAS) networks. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 776–789.

[62] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. 2020. Adversarial weight perturbation helps robust generalization. *Advances in Neural Information Processing Systems* 33 (2020), 2958–2969.

[63] Yannan Nellie Wu, Po-An Tsai, Saurav Muralidharan, Angshuman Parashar, Vivienne Sze, and Joel Emer. 2023. HighLight: Efficient and Flexible DNN Acceleration with Hierarchical Structured Sparsity. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 1106–1120.

[64] Chang Xiao, Peilin Zhong, and Changxi Zheng. 2020. Enhancing Adversarial Defense by k-Winners-Take-All. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Skgvy64tvr

[65] Fuxun Yu, Zhuwei Qin, and Xiang Chen. 2018. Distilling critical paths in convolutional neural networks. *arXiv preprint arXiv:1811.02643* (2018).

[66] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146* (2016).

[67] Guowei Zhang, Nithya Attaluri, Joel S Emer, and Daniel Sanchez. 2021. Gamma: Leveraging Gustavson's algorithm to accelerate sparse matrix multiplication. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 687–701.