The Shape of Noise: Layer-Wise Perturbation Profiles for Diagnosing Vision Robustness

Son Nguyen [1] **V. G. Bao** [2] **Quang Minh Phan** [3] **Trong P. Le** [2]

Abstract

As vision models are increasingly applied across diverse applications, the need for explainable and robust architectures continues to grow. Existing corruption robustness benchmarks, such as CIFAR-10-C, reduce model behavior to aggregate metrics like mean Corruption Error, obscuring how perturbations are amplified, suppressed, or transformed within a network. We introduce the Perturbation Evaluation Framework (PEF), a layer-wise diagnostic framework that decomposes a model's response to input corruption and reveals architecture-dependent amplification and suppression signatures. Through experiments including intermediate-layer perturbation injection and profile-guided stabilization, we demonstrate that PEF complements aggregate benchmarks by providing interpretable layer-wise diagnostics for analyzing and targeting robustness behavior.

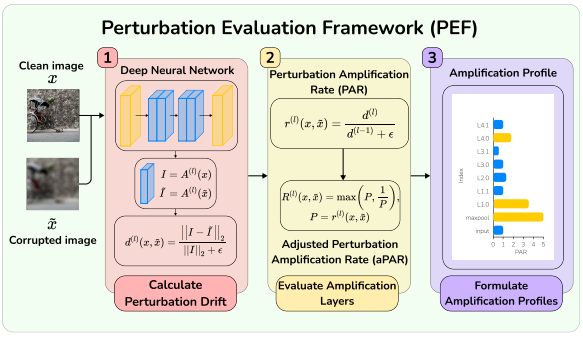


Figure 1. Structure of PEF. The framework operates in three stages: (1) calculating RRD—the perturbation drift between clean and corrupted inputs—at each layer, (2) calculating PAR and aPAR at each layer, and (3) formulating amplification profile from layer-wise PARs.

1. Introduction

As computer vision is increasingly applied to real-world problems, vision models are frequently exposed to natural corruptions. To address these reliability concerns, researchers have proposed diverse strategies, including advanced data augmentation techniques like AugMix and RandAugment (Hendrycks et al., 2020; Cubuk et al., 2020), architectural modifications such as anti-aliasing filters (Zhang, 2019) and self-attention mechanisms (Bhojanapalli et al., 2021), as well as specialized training procedures like Noisy Student (Xie et al., 2020) and adversarial noise training (Rusak et al., 2020). Existing corruption benchmarks, including CIFAR-10-C and ImageNet-C (Hendrycks & Dietterich, 2019), have been central to measuring the reliability of these defenses by reporting aggregate performance degradation under standardized corruptions. However, these scalar scores primarily answer whether a model is robust, but yield minimal information about how corruption-induced perturbations propagate through the network, which layers amplify or suppress them, and whether different architectures fail through the same internal mechanisms.

This missing internal view matters for both diagnosis and intervention. A corruption may be introduced only at the input, but its effect can be transformed non-uniformly across depth: some layers may suppress the incoming representation drift, others may preserve it, and others may amplify it. Since these internal dynamics are reduced to a single performance scalar, aggregate robustness metrics cannot distinguish these cases, and provide limited guidance for identifying architecture-dependent failure modes (Geirhos et al., 2020), comparing how different model families process perturbations (Raghu et al., 2021), or deciding where targeted adaptation may be more efficient than updating an entire network (Schneider et al., 2020).

Therefore, we argue that aggregate robustness evaluation should be complemented by layer-wise propagation diagnostics that satisfy three desiderata: **(1) Quantitative decomposability:** separating cumulative representation drift from each layer's local contribution; **(2) Mechanistic localization:** identifying which components amplify, suppress, or preserve corruption-induced drift; **(3) Actionable layer se-**

[1]Arizona State University [2]VNUHCM—University of Information Technology [3]University of Minnesota—Twin Cities. Correspondence to: Son Nguyen <snguye88@asu.edu>.

Accepted to the 1st Workshop on Combining Theory and Benchmarks, CTB@ICML 2026, Seoul, South Korea.

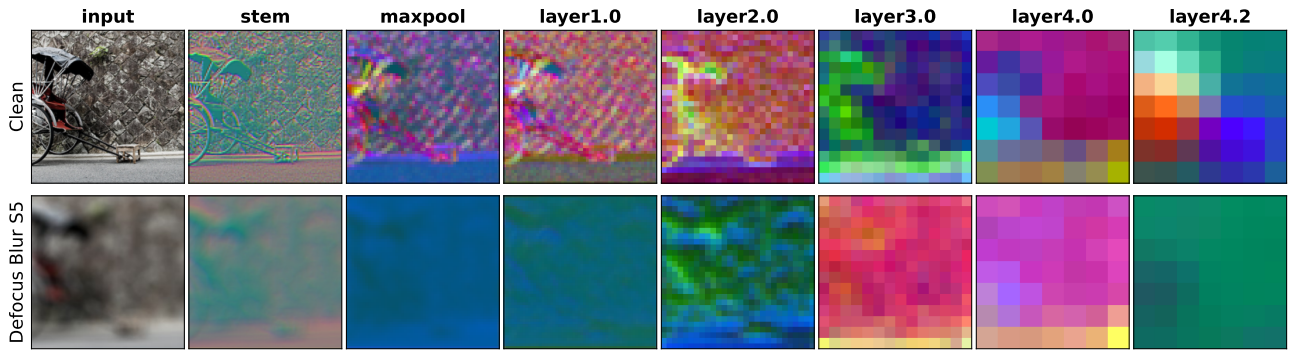


Figure 2. Visualizing corruption-induced representation drift. Clean and corrupted inputs cause different representation shifts across depth. Although corruption is introduced only at the input, its effect is transformed non-uniformly across layers, motivating layer-wise robustness diagnostics rather than relying only on aggregate output accuracy.

lection: providing candidate regions for parameter-efficient robustness interventions, while remaining explicit that such candidates are diagnostic hypotheses rather than guaranteed optimal adaptation targets.

Contributions. We introduce the **Perturbation Evaluation Framework (PEF)** (Figure 1), a layer-wise framework for diagnosing corruption-induced representation drift. Our main contributions are:

1. We provide a framework that measures downstream representation drift and layer-wise perturbation changes, separating global accumulation and local changes.

2. We empirically show that amplification profiles are architecture-dependent—distinct models process and distribute perturbations differently.

3. Using intermediate-layer activation noise, we test whether amplification profiles reveal functionally sensitive regions, finding that noise-injected non-neutral layers damage model robustness more than noise-injected near-neutral layers.

4. Through evaluation, we find that targeting non-neutral layers for robustness intervention retains much of the accuracy of broader adaptation while being more parameter-efficient.

2. Method

2.1. Setup

Let $\mathbf{x}$ denote a clean image and $\tilde{\mathbf{x}}$ its corrupted counterpart. We pass both inputs through the same network and let $A^{(l)}(\cdot)$ denote the activation tensor at layer l. PEF tracks how the representation difference between $\mathbf{x}$ and $\tilde{\mathbf{x}}$ evolves across layers, allowing us to distinguish cumulative perturbation drift from local layer-wise amplification or suppression (see Appendix C.1 for more details).

2.2. Relative Representation Difference (RRD)

For each layer l, we define the **Relative Representation Difference** as:

$$d^{(l)}(\mathbf{x}, \tilde{\mathbf{x}}) = \frac{\|A^{(l)}(\mathbf{x}) - A^{(l)}(\tilde{\mathbf{x}})\|_2}{\|A^{(l)}(\mathbf{x})\|_2 + \epsilon} \tag{1}$$

where ϵ denotes a small numerical constant that prevents division by zero, and $\|\cdot\|_2$ represents the L_2 norm.

This quantity measures the difference at layer l between the original and the corrupted input's representation, i.e., the total perturbation-induced drift that has accumulated by layer l, normalized by the representation size.

2.3. Perturbation Amplification Rate (PAR)

To quantify how each layer locally alters the perturbation propagating from the previous layer, we introduce the **Perturbation Amplification Rate**—for layer l:

$$r^{(l)}(\mathbf{x}, \tilde{\mathbf{x}}) = \frac{d^{(l)}(\mathbf{x}, \tilde{\mathbf{x}})}{d^{(l-1)}(\mathbf{x}, \tilde{\mathbf{x}}) + \epsilon} \tag{2}$$

where ϵ denotes a small numerical constant that prevents division by zero.

Values above 1 indicate local amplification, values near 1 indicate approximately neutral propagation, and values below 1 indicate suppression. These three regimes demonstrate that PAR acts as an empirical local scaling factor, defining both the direction and magnitude of the perturbation drift.

2.4. Selecting candidate layers for targeted intervention

For targeted intervention, both strong amplification and suppression may indicate non-neutral perturbation processing.

We therefore define a direction-agnostic score, **adjusted Perturbation Amplification Rate (aPAR)**:

$$R^{(l)} = \max\left(\bar{r}^{(l)}, \frac{1}{\bar{r}^{(l)}}\right) \tag{3}$$

where $\bar{r}^{(l)}$ denotes mean PAR over images, corruption types, and severity levels.

3. Experiments

3.1. Setup

Models. We evaluate PEF on ConvNeXt-Tiny (Liu et al., 2022) and ResNet-50 (He et al., 2016), two CNN architectures with different stage structures, downsampling patterns, and residual/block designs (see Appendix B.1 for more details).

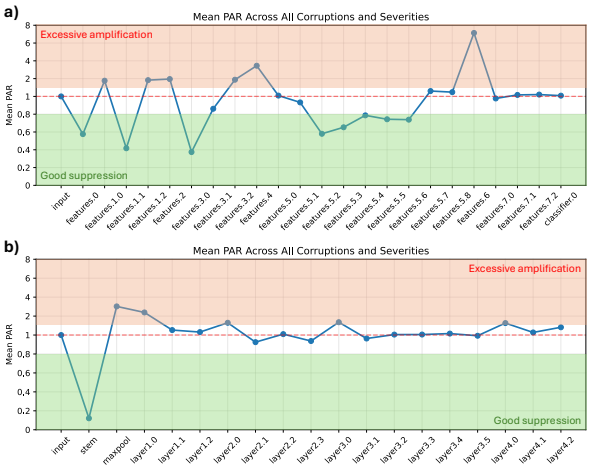


Figure 3. Amplification profiles across all CIFAR-10-C corruptions and severities for (a) ConvNeXt-Tiny and (b) ResNet-50. The shaded regions denote structural regimes of high perturbation amplification (red) and high perturbation suppression (green).

Datasets. Amplification profiles are constructed from paired CIFAR-10 (Krizhevsky, 2009) images and corrupted variants following the CIFAR-10-C corruption taxonomy. For targeted stabilization, we split the CIFAR-10 training set into an 85% training subset and a 15% held-out validation subset. Adaptation methods are trained on the 85% subset, clean validation accuracy is reported on the held-out 15% subset, and corruption robustness is evaluated on CIFAR-10-C (see Appendix B.2 for more details).

Evaluation overview. We ask three questions: **(1)** Do different architectures exhibit different perturbation amplification profiles? **(2)** Do non-neutral layers derived from amplification profiles correspond to functionally sensitive regions under controlled internal perturbation? **(3)** To what extent can these layers serve as viable candidates for parameter-efficient robustness interventions?

3.2. Amplification profiles across architectures

A model's amplification profile consists of the layer-wise PAR averaged across images, corruption types, and severity levels (see Appendix C.2 for more details).

Figure 3 shows that the two architectures exhibit qualitatively different amplification profiles, suggesting that robustness failures can be architecture-dependent even under the same corruption benchmark.

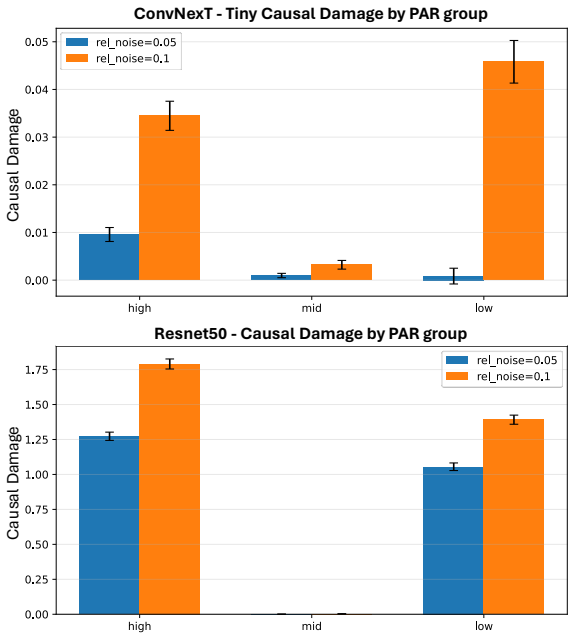


Figure 4. Causal damage from norm-controlled internal noise injection grouped by PAR regime. Near-neutral layers are less damaging, while the most harmful non-neutral direction differs across architectures.

ConvNeXt-Tiny. ConvNeXt-Tiny exhibits a highly fluctuating amplification profile. Early stages oscillate sharply (`features.0` through `features.3.0`), before transitioning to a notable mid-network spike (`features.3.2` and `features.4`). However, later stages feature a prolonged suppression block (`features.5.2` through `features.5.6`), until the largest amplification emerges deep in the network (`features.5.8`). Finally, the last stage blocks (`features.7.x`) stabilize the noise.

ResNet-50. In contrast, ResNet-50's amplification profile is heavily front-loaded. After a significant initial suppression (`stem`), two consecutive amplification peaks occur. Conversely, subsequent residual blocks restore stability, hovering near a neutral PAR of 1.0.

Table 1. Targeted stabilization results comparing baseline models, full fine-tuning, and parameter-efficient methods (adapters, LoRA) applied to different layer subsets (top-k, bottom-k, random-k). Results show Clean and CIFAR-10-C accuracy on the subset of 3 corruption types, along with trainable parameter counts for ConvNeXt-Tiny and ResNet-50.

Method	ConvNeXt-Tiny			ResNet-50		
	Clean acc. ↑	CIFAR-10-C subset acc. ↑	Parameters (M) ↓	Clean acc. ↑	CIFAR-10-C subset acc. ↑	Parameters (M) ↓
Pretrained	68.29% ± 0.00%	49.57% ± 0.30%	0.00 ± 0.00	54.64% ± 0.00%	35.16% ± 1.45%	0.00 ± 0.00
Full fine-tune	97.58% ± 0.41%	94.95% ± 1.23%	27.83 ± 0.00	93.77% ± 2.22%	91.53% ± 2.45%	23.53 ± 0.00
Adapter (top-k)	90.89% ± 0.50%	90.38% ± 5.24%	1.18 ± 0.00	70.65% ± 2.73%	86.02% ± 7.65%	8.38 ± 0.00
Adapter (bottom-k)	90.09% ± 1.89%	91.84% ± 2.37%	2.03 ± 0.03	58.35% ± 18.52%	85.70% ± 7.43%	4.54 ± 0.83
Adapter (random-k)	91.33% ± 0.55%	90.57% ± 5.32%	1.14 ± 0.59	71.23% ± 5.79%	84.96% ± 7.16%	6.90 ± 2.50
LoRA (top-k)	95.10% ± 1.98%	92.57% ± 2.43%	0.04 ± 0.00	92.72% ± 3.51%	89.93% ± 3.81%	0.20 ± 0.00
LoRA (bottom-k)	94.75% ± 1.94%	91.90% ± 2.09%	0.83 ± 0.03	91.67% ± 3.82%	88.65% ± 3.79%	0.15 ± 0.01
LoRA (random-k)	94.36% ± 3.08%	91.61% ± 3.35%	0.37 ± 0.04	90.60% ± 4.06%	87.12% ± 4.69%	0.17 ± 0.01

Key insights. While ConvNeXt-Tiny shows distributed non-neutral behavior across depth, ResNet-50 is dominated by early-layer variation followed by near-neutral propagation. These two architectures exhibit qualitatively distinct amplification profiles—behaviors and suppression capacities—throughout layers, which aggregate benchmarking cannot reflect. This highlights the need for layer-aware evaluation frameworks.

3.3. Internal causal injection

Method. To test whether amplification profiles correspond to functional sensitivity, we perform norm-controlled internal noise injection. For a selected layer, we add Gaussian noise to the clean activation tensor, scaled relative to the activation norm, and then continue the forward pass through the remaining layers. We define causal damage as the resulting drop in downstream accuracy relative to the unperturbed clean forward pass (see Appendix E for more details).

Results. Figure 4 shows that PAR regimes are functionally informative but architecture-dependent. In both models, extreme-PAR layers induce substantially more degradation than near-neutral layers. However, the primary vulnerabilities vary: ConvNeXt-Tiny is most sensitive to perturbations in low-PAR layers, whereas ResNet-50 degrades most when high-PAR extremes are perturbed. This indicates that amplification profiles can help localize robustness-relevant layers, but the mapping from perturbation profiles to causal damage is not universal. This motivates aPAR as a direction-agnostic first-order heuristic to identify such layers.

3.4. Targeted intervention

Method. We next evaluate whether amplification profiles can guide parameter-efficient robustness intervention. We compare targeted updates on the **top-k** aPAR layers against the bottom-k and random-k selections, as well as against full fine-tuning. We consider three intervention methods: direct fine-tuning (Yosinski et al., 2014), adapters (Rebuffi et al., 2017), and LoRA (Hu et al., 2022). These comparisons test whether profile-selected non-neutral layers provide a useful alternative to uniformly adapting the full model (see Appendix D for more details).

Results. Table 1 shows that profile-guided stabilization recovers much of full fine-tuning's robustness while using substantially fewer parameters. For LoRA, high-aPAR selection consistently beats random baselines. Conversely, adapter results are more nuanced: lower-aPAR layers often perform better. This demonstrates that aPAR is a diagnostic heuristic, not a formal guarantee, and optimal intervention targets depend on the chosen adaptation method.

4. Discussion and Conclusion

We have presented PEF, a framework that turns scalar robustness scores into layer-wise amplification profiles and identifies candidate layers for stabilization through deviations from neutral propagation.

Findings. ConvNeXt-Tiny and ResNet-50 exhibit different perturbation dynamics: ConvNeXt-Tiny shows distributed fluctuations, while ResNet-50 concentrates variation in early stages. Injection and intervention experiments suggest that non-neutral layers are useful robustness targets, though the most effective intervention depends on architecture and method.

Implications. PEF suggests that corruption robustness should be evaluated not only as an aggregate output metric, but also as a layer-wise propagation profile. Depending on the interplay between methods and architectures, such profiles can potentially reveal architecture-dependent failure patterns, help generate testable hypotheses about where robustness bottlenecks arise, and suggest targeted intervention strategies.

Limitations and future directions. Future work should extend PEF to diverse architectures (e.g., Vision Transformers), more complex datasets, natural distribution shifts, and broader intervention families. Another important direction is to better characterize when PAR and aPAR predict downstream robustness, both empirically and theoretically.

References

Bhojanapalli, S., Chakrabarti, A., Glasner, D., Li, D., Unterthiner, T., and Veit, A. Understanding robustness of transformers for image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10231–10241, 2021.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. RandAugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3008–3017, 2020.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. IEEE, 2009.

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Hendrycks, D. and Dietterich, T. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019.

Hendrycks, D., Mu, N., Cubuk, E. D., Zoph, B., Gilmer, J., and Lakshminarayanan, B. AugMix: A simple data processing method to improve robustness and uncertainty. In *International Conference on Learning Representations*, 2020.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. A ConvNet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11976–11986, 2022.

Raghu, M., Unterthiner, T., Kornblith, S., Zhang, C., and Dosovitskiy, A. Do vision transformers see like convolutional neural networks? In *Advances in Neural Information Processing Systems*, volume 34, pp. 12116–12128, 2021.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Rusak, E., Schott, L., Zimmermann, R. S., Bitterwolf, J., Bringmann, O., Bethge, M., and Brendel, W. A simple way to make neural networks robust against diverse image corruptions. In *European Conference on Computer Vision*, pp. 53–69. Springer, 2020.

Schneider, S., Rusak, E., Eck, L., Bringmann, O., Brendel, W., and Bethge, M. Improving robustness against common corruptions by covariate shift adaptation. In *Advances in Neural Information Processing Systems*, volume 33, pp. 11539–11551, 2020.

Xie, Q., Luong, M.-T., Hovy, E., and Le, Q. V. Self-training with noisy student improves ImageNet classification, robustness, and adversarial robustness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10687–10698, 2020.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, volume 27, 2014.

Zhang, R. Making convolutional networks shift-invariant again. In *International Conference on Machine Learning*. PMLR, 2019.

A. Formalizing PEF as a structured benchmark diagnostic

PEF treats a corruption benchmark not only as an output-level accuracy test, but also as a distribution over clean–corrupted representation pairs. Let $Z = (x, c, s)$ denote a benchmark sample drawn from a corruption benchmark distribution $\mathcal{Q}$, where x is a clean image, c is a corruption type, and s is a severity level. Let $\tilde{x}_{c,s}$ denote the corrupted counterpart of x under corruption c and severity s.

For each layer $l \in \{0, \ldots, L\}$, let $z^{(l)}(x)$ and $z^{(l)}(\tilde{x}_{c,s})$ denote the representations used for RRD computation. These may be full activation tensors or pooled/flattened representations, depending on the implementation. We define the Relative Representation Difference (RRD) at layer l as

$$d^{(l)}(Z) = \frac{\left\| z^{(l)}(x) - z^{(l)}(\tilde{x}_{c,s}) \right\|_2}{\left\| z^{(l)}(x) \right\|_2 + \epsilon}, \tag{4}$$

where $\epsilon > 0$ stabilizes the clean-representation normalization.

For numerical stability in PAR, we use a separate positive constant $\tau > 0$ and define the regularized drift

$$d_\tau^{(l)}(Z) = d^{(l)}(Z) + \tau. \tag{5}$$

The regularized Perturbation Amplification Rate (PAR) is then

$$r_\tau^{(l)}(Z) = \frac{d_\tau^{(l)}(Z)}{d_\tau^{(l-1)}(Z)}, \qquad l = 1, \ldots, L. \tag{6}$$

The following properties formalize PEF as a layer-wise benchmark diagnostic. They establish that PAR decomposes cumulative representation drift into local multiplicative factors and that aPAR provides a symmetric score for deviations from neutral propagation. These results concern the diagnostic quantities themselves and should not be interpreted as guarantees of downstream adaptation performance.

Proposition A.1 (Local propagation regimes)**.** *For any benchmark sample Z and layer $l \in \{1, \ldots, L\}$,*

$$d_\tau^{(l)}(Z) = r_\tau^{(l)}(Z) d_\tau^{(l-1)}(Z). \tag{7}$$

Moreover, because the same positive constant τ is added to both drift values,

$$d^{(l)}(Z) > d^{(l-1)}(Z) \iff r_\tau^{(l)}(Z) > 1, \tag{8}$$

$$d^{(l)}(Z) = d^{(l-1)}(Z) \iff r_\tau^{(l)}(Z) = 1, \tag{9}$$

$$d^{(l)}(Z) < d^{(l-1)}(Z) \iff r_\tau^{(l)}(Z) < 1. \tag{10}$$

Thus PAR partitions local drift propagation into amplification, neutral propagation, and suppression regimes.

Proof. The identity follows directly from the definition of $r_\tau^{(l)}(Z)$:

$$r_\tau^{(l)}(Z) = \frac{d_\tau^{(l)}(Z)}{d_\tau^{(l-1)}(Z)}.$$

Multiplying both sides by the positive denominator $d_\tau^{(l-1)}(Z)$ yields

$$d_\tau^{(l)}(Z) = r_\tau^{(l)}(Z) d_\tau^{(l-1)}(Z).$$

Since $\tau > 0$ is added to both $d^{(l)}(Z)$ and $d^{(l-1)}(Z)$, the ordering between the two drift values is unchanged:

$$d^{(l)}(Z) \gtrless d^{(l-1)}(Z) \iff d_\tau^{(l)}(Z) \gtrless d_\tau^{(l-1)}(Z).$$

Dividing by the positive quantity $d_\tau^{(l-1)}(Z)$ gives the stated equivalences. □

Proposition A.2 (Multiplicative decomposition across depth)**.** *For any benchmark sample Z and any two layers $0 \le a < b \le L$,*

$$\frac{d_\tau^{(b)}(Z)}{d_\tau^{(a)}(Z)} = \prod_{l=a+1}^{b} r_\tau^{(l)}(Z). \tag{11}$$

Equivalently,

$$\log d_\tau^{(b)}(Z) - \log d_\tau^{(a)}(Z) = \sum_{l=a+1}^{b} \log r_\tau^{(l)}(Z). \tag{12}$$

Proof. By Proposition A.1,

$$d_\tau^{(l)}(Z) = r_\tau^{(l)}(Z) d_\tau^{(l-1)}(Z)$$

for each l. Repeatedly applying this identity from $l = a+1$ to $l = b$ gives

$$d_\tau^{(b)}(Z) = \left(\prod_{l=a+1}^{b} r_\tau^{(l)}(Z) \right) d_\tau^{(a)}(Z).$$

Dividing by $d_\tau^{(a)}(Z) > 0$ yields the product identity. Taking logarithms gives the additive log-space decomposition. □

Proposition A.3 (Layer-transition factorization of PAR)**.** *For a fixed benchmark sample Z, suppose $d^{(l-1)}(Z) > 0$ and define the unregularized PAR*

$$r^{(l)}(Z) = \frac{d^{(l)}(Z)}{d^{(l-1)}(Z)}.$$

Using the shorthand

$$z^{(l)} = z^{(l)}(x), \qquad \tilde{z}^{(l)} = z^{(l)}(\tilde{x}_{c,s}),$$

the unregularized PAR admits the exact factorization

$$r^{(l)}(Z) = \underbrace{\frac{\left\| z^{(l)} - \tilde{z}^{(l)} \right\|_2}{\left\| z^{(l-1)} - \tilde{z}^{(l-1)} \right\|_2}}_{\text{empirical local expansion}} \cdot \underbrace{\frac{\left\| z^{(l-1)} \right\|_2 + \epsilon}{\left\| z^{(l)} \right\|_2 + \epsilon}}_{\text{clean-norm renormalization}}. \tag{13}$$

If the measured transition can be represented by a map G_l such that $z^{(l)} = G_l(z^{(l-1)})$ and $\tilde{z}^{(l)} = G_l(\tilde{z}^{(l-1)})$, and if G_l is L_l-Lipschitz on the evaluated domain, then

$$r^{(l)}(Z) \le L_l \frac{\left\| z^{(l-1)} \right\|_2 + \epsilon}{\left\| z^{(l)} \right\|_2 + \epsilon}. \tag{14}$$

Proof. Substituting the definition of RRD into $r^{(l)}(Z) = d^{(l)}(Z)/d^{(l-1)}(Z)$ gives

$$r^{(l)}(Z) = \frac{\left\| z^{(l)} - \tilde{z}^{(l)} \right\|_2}{\left\| z^{(l)} \right\|_2 + \epsilon} \cdot \frac{\left\| z^{(l-1)} \right\|_2 + \epsilon}{\left\| z^{(l-1)} - \tilde{z}^{(l-1)} \right\|_2}.$$

Rearranging terms yields the factorization in Equation 13. If the measured transition admits a map G_l and G_l is L_l-Lipschitz, then

$$\left\| z^{(l)} - \tilde{z}^{(l)} \right\|_2 = \left\| G_l(z^{(l-1)}) - G_l(\tilde{z}^{(l-1)}) \right\|_2 \le L_l \left\| z^{(l-1)} - \tilde{z}^{(l-1)} \right\|_2.$$

Substituting this inequality into Equation 13 gives the stated upper bound. □

Proposition A.4 (Symmetric multiplicative deviation)**.** *For any $r > 0$, define the adjusted Perturbation Amplification Rate*

$$R(r) = \max\left(r, \frac{1}{r} \right).$$

Then

$$R(r) = \exp\left(|\log r| \right). \tag{15}$$

Consequently, $R(r) = R(1/r)$, $R(r) \ge 1$ with equality if and only if $r = 1$, and $R(r)$ increases monotonically with the absolute log-deviation $|\log r|$ from neutral propagation.

Proof. If $r \geq 1$, then $|\log r| = \log r$, so

$$\exp(|\log r|) = \exp(\log r) = r.$$

If $0 < r < 1$, then $|\log r| = -\log r = \log(1/r)$, so

$$\exp(|\log r|) = \exp(\log(1/r)) = \frac{1}{r}.$$

Therefore

$$\exp(|\log r|) = \max\left(r, \frac{1}{r}\right).$$

The symmetry $R(r) = R(1/r)$ follows from $|\log r| = |\log(1/r)|$. The lower bound $R(r) \geq 1$ follows because $\max(r, 1/r) \geq 1$ for all $r > 0$, with equality only when $r = 1$. Finally, because $\exp(\cdot)$ is strictly increasing, $R(r) = \exp(|\log r|)$ increases monotonically with $|\log r|$. □

Motivated by Proposition A.4, we define the pointwise log-aPAR score at layer l as

$$S^{(l)}(Z) = \left|\log r_\tau^{(l)}(Z)\right| = \log \max\left(r_\tau^{(l)}(Z), \frac{1}{r_\tau^{(l)}(Z)}\right). \tag{16}$$

This score is zero at neutral propagation and increases symmetrically under multiplicative amplification or suppression. The population PEF profile under benchmark distribution $\mathcal{Q}$ is

$$\mu_l = \mathbb{E}_{Z\sim\mathcal{Q}}\left[S^{(l)}(Z)\right], \qquad l = 1, \ldots, L. \tag{17}$$

Given n benchmark samples $Z_1, \ldots, Z_n$, the empirical PEF profile is

$$\widehat{\mu}_l = \frac{1}{n}\sum_{i=1}^{n} S^{(l)}(Z_i). \tag{18}$$

The profile can be reported on the log scale as $\widehat{\mu}_l$ or on the multiplicative aPAR scale as $\exp(\widehat{\mu}_l)$; these induce the same ranking over layers.

Proposition A.5 (Uniform concentration of estimated PEF profiles)**.** *Assume that the log-aPAR scores are clipped or bounded so that $S^{(l)}(Z) \in [0, M]$ for all layers l and benchmark samples Z. Let $Z_1, \ldots, Z_n$ be independent samples from $\mathcal{Q}$. Then, with probability at least $1 - \eta$,*

$$\max_{1\leq l\leq L} |\widehat{\mu}_l - \mu_l| \leq M\sqrt{\frac{\log(2L/\eta)}{2n}}. \tag{19}$$

Proof. For a fixed layer l, Hoeffding's inequality gives

$$\Pr\left(|\widehat{\mu}_l - \mu_l| \geq t\right) \leq 2\exp\left(-\frac{2nt^2}{M^2}\right),$$

because $S^{(l)}(Z) \in [0, M]$. Applying a union bound over all L layers gives

$$\Pr\left(\max_{1\leq l\leq L} |\widehat{\mu}_l - \mu_l| \geq t\right) \leq 2L\exp\left(-\frac{2nt^2}{M^2}\right).$$

Setting the right-hand side equal to η and solving for t gives

$$t = M\sqrt{\frac{\log(2L/\eta)}{2n}}.$$

Thus, with probability at least $1 - \eta$,

$$\max_{1\leq l\leq L} |\widehat{\mu}_l - \mu_l| \leq M\sqrt{\frac{\log(2L/\eta)}{2n}}.$$

□

Corollary A.6 (Stability of empirical top-k selection)**.** *Let $\mu_{(1)} \geq \mu_{(2)} \geq \cdots \geq \mu_{(L)}$ denote the sorted population PEF profile scores. Define*

$$\rho = M\sqrt{\frac{\log(2L/\eta)}{2n}}.$$

If the population gap satisfies

$$\mu_{(k)} - \mu_{(k+1)} > 2\rho, \tag{20}$$

then the empirical top-k layers selected by $\widehat{\mu}_l$ recover the population top-k layers with probability at least $1-\eta$.

Proof. By Proposition A.5, with probability at least $1-\eta$, every empirical score is within ρ of its population score:

$$|\widehat{\mu}_l - \mu_l| \leq \rho \qquad \text{for all } l.$$

Consider any population top-k layer i and any non-top-k layer j. By the gap condition,

$$\mu_i - \mu_j \geq \mu_{(k)} - \mu_{(k+1)} > 2\rho.$$

Therefore,

$$\widehat{\mu}_i \geq \mu_i - \rho > \mu_j + \rho \geq \widehat{\mu}_j.$$

Thus every population top-k layer has empirical score larger than every non-top-k layer. Hence the empirical top-k set equals the population top-k set with probability at least $1-\eta$. □

Proposition A.7 (Top-k optimality for diagnostic coverage)**.** *Let $q_l \geq 0$ be a fixed diagnostic score for layer l, such as the population score μ_l or empirical score $\widehat{\mu}_l$. For a layer-count budget k, define the diagnostic coverage of a selected subset $\mathcal{K}$ as*

$$I(\mathcal{K}) = \sum_{l \in \mathcal{K}} q_l, \qquad |\mathcal{K}| = k. \tag{21}$$

Then $I(\mathcal{K})$ is maximized by selecting the k layers with largest values of q_l.

Proof. Sort the layers so that

$$q_{l_1} \geq q_{l_2} \geq \cdots \geq q_{l_L}.$$

Let

$$\mathcal{K}_{\text{top}} = \{l_1, \ldots, l_k\}$$

be the top-k set. Consider any other subset $\mathcal{K}$ with $|\mathcal{K}| = k$. If $\mathcal{K} = \mathcal{K}_{\text{top}}$, the result is immediate. Otherwise, there exists at least one layer $i \in \mathcal{K}_{\text{top}} \setminus \mathcal{K}$ and at least one layer $j \in \mathcal{K} \setminus \mathcal{K}_{\text{top}}$. By construction, $q_i \geq q_j$. Replacing j with i does not decrease the objective $I(\mathcal{K})$. Repeating this exchange transforms $\mathcal{K}$ into $\mathcal{K}_{\text{top}}$ without decreasing the objective. Therefore $\mathcal{K}_{\text{top}}$ maximizes $I(\mathcal{K})$. □

Proposition A.7 establishes optimality only for the additive diagnostic coverage objective in Equation 21. It does not imply that the selected layers are universally optimal for downstream adaptation, since intervention effects may be non-additive and different layers may incur different trainable parameter costs. If layers have unequal adaptation costs p_l, then a parameter-budgeted diagnostic selection problem is instead

$$\max_{\mathcal{K}} \sum_{l \in \mathcal{K}} q_l \qquad \text{subject to} \qquad \sum_{l \in \mathcal{K}} p_l \leq B, \tag{22}$$

which is a cost-constrained subset selection problem. In this work, top-k aPAR selection is therefore used as a diagnostic layer-selection heuristic: it targets layers with the largest estimated deviations from neutral perturbation propagation, while leaving the effectiveness of downstream adaptation to empirical validation.

B. Experimental setup

B.1. Architectures

B.1.1. ConvNeXt-Tiny

We use the ImageNet-pretrained ConvNeXt-Tiny (Liu et al., 2022) backbone from `torchvision`, replacing the original classification layer with a newly initialized 10-class linear head. ConvNeXt-Tiny begins with a non-overlapping 4×4 patchifying convolution with stride 4, followed by Layer Normalization. The network then applies four ConvNeXt stages containing 3, 3, 9, and 3 blocks, respectively. Each block uses a 7×7 depthwise convolution, an inverted bottleneck implemented with pointwise linear projections, GELU activation, layer-scale parameters, and a residual connection. Downsampling between stages is handled by separate downsampling modules rather than by maxpooling.

For PEF diagnosis, we may record activations at the input, patchifying stem, downsampling modules, and ConvNeXt blocks in model order. For targeted intervention, however, we restrict the targetable ConvNeXt-Tiny modules to ConvNeXt blocks only. The stem and downsampling modules are therefore used for diagnosis when relevant, but are not used as trainable intervention targets.

B.1.2. ResNet-50

We use the ImageNet-pretrained ResNet-50 (He et al., 2016) backbone from `torchvision`, replacing the original classification layer with a newly initialized 10-class linear head. ResNet-50 consists of an initial convolutional stem followed by maxpooling and four residual stages, `layer1` to `layer4`, containing 3, 4, 6, and 3 bottleneck residual blocks, respectively. Each bottleneck block applies a 1×1 channel-reduction convolution, a 3×3 spatial convolution, and a 1×1 channel-expansion convolution, together with a residual skip connection. Spatial downsampling occurs at the beginning of later residual stages.

For PEF diagnosis, we may record activations at the input, stem, maxpooling layer, and residual blocks in model order. For targeted intervention, however, we restrict the targetable ResNet-50 modules to residual blocks only. The input, convolutional stem, and maxpooling layers are therefore used for diagnosis when relevant, but are not used as trainable intervention targets.

Stage	Targetable ResNet-50 modules
`layer1`	`layer1.0`, `layer1.1`, `layer1.2`
`layer2`	`layer2.0`, `layer2.1`, `layer2.2`, `layer2.3`
`layer3`	`layer3.0`, `layer3.1`, `layer3.2`, `layer3.3`, `layer3.4`, `layer3.5`
`layer4`	`layer4.0`, `layer4.1`, `layer4.2`

Stage	Targetable ConvNeXt-Tiny modules
`features.1`	`features.1.0`, `features.1.1`, `features.1.2`
`features.3`	`features.3.0`, `features.3.1`, `features.3.2`
`features.5`	`features.5.0`, `features.5.1`, `features.5.2`, `features.5.3`, `features.5.4`, `features.5.5`, `features.5.6`, `features.5.7`, `features.5.8`
`features.7`	`features.7.0`, `features.7.1`, `features.7.2`

B.2. Dataset and corruption protocol

B.2.1. Base dataset

We use CIFAR-10 (Krizhevsky, 2009) as the base classification dataset. The original CIFAR-10 training split contains 50,000 images. We partition this split into 85% training data and 15% validation data using a fixed stratified split controlled by the run's global seed (evaluated on seeds 42 and 43), yielding 42,500 training images and 7,500 validation images. The same split is reused for every architecture, intervention method, and learning-rate trial.

All images are resized to 224×224 using bicubic interpolation and normalized with ImageNet (Deng et al., 2009) channel statistics:

$$\mu = (0.485, 0.456, 0.406), \qquad \sigma = (0.229, 0.224, 0.225).$$

This preprocessing is shared by clean training, corrupted training, validation, aPAR discovery, and final evaluation.

B.2.2. CORRUPTION FAMILY

We follow the common-corruption taxonomy of CIFAR-10-C / ImageNet-C (Hendrycks & Dietterich, 2019). The full corruption list used by the implementation is:

`gaussian_noise`, `shot_noise`, `impulse_noise`, `defocus_blur`, `glass_blur`, `motion_blur`, `zoom_blur`, `snow`, `frost`, `fog`, `brightness`, `contrast`, `elastic_transform`, `pixelate`, `jpeg_compression`, `speckle_noise`, `gaussian_blur`, `saturate`, `spatter`.

For training, validation, and aPAR-based layer selection, we sample 3 active corruption types once per global seed and keep this subset fixed across all methods for that run. For instance, using seed 42, the selected zero-indexed corruption IDs are $[15, 18, 17]$, corresponding to:

$$\mathcal{C}_{\text{active}} = \{\texttt{speckle\_noise}, \texttt{spatter}, \texttt{saturate}\}.$$

Training and validation use severity level 5 only. Thus, the active training condition set is

$$\mathcal{S}_{\text{train}} = \{(\texttt{speckle\_noise}, 5), (\texttt{spatter}, 5), (\texttt{saturate}, 5)\}.$$

During training and validation, corrupted images are generated deterministically on the fly from the clean CIFAR-10 images. For a fixed seed, split identifier, epoch index, sample index, corruption type, and severity, the same corrupted image is generated. The training dataset uses a balanced epoch schedule: each epoch contains all active corruption types, with the corruption assignment rotated deterministically across epochs. Therefore, no epoch is exclusively associated with a single corruption type.

For final testing, we use the pre-generated CIFAR-10-C data through the HuggingFace dataset `randall-lab/cifar10-c`. The final test evaluates the fixed active corruption subset $\mathcal{C}_{\text{active}}$ across all severity levels 1–5. Unless otherwise stated, reported CIFAR-10-C test accuracy in the intervention experiments therefore refers to this fixed CIFAR-10-C corruption subset rather than the full 19-corruption benchmark.

B.3. Implementation and deterministic configuration

All experiments use a deterministic data-generation protocol tied to a specific global seed. For a given run, the implementation uniformly seeds Python, NumPy, PyTorch CPU, and PyTorch CUDA randomness; it enables deterministic cuDNN behavior, and disables cuDNN benchmarking and TF32 matrix multiplication. Below is an example with seed 42:

```
os.environ["PYTHONHASHSEED"] = "42"
random.seed(42)
np.random.seed(42)
torch.manual_seed(42)
torch.cuda.manual_seed_all(42)
torch.backends.cudnn.benchmark = False
torch.backends.cudnn.allow_tf32 = False
torch.backends.cuda.matmul.allow_tf32 = False
```

DataLoader shuffling is controlled by an explicitly seeded CPU generator. Corruption randomness is generated from stable hash-based seeds that depend on the sample, epoch, split, corruption identity, and severity (refer to Table 2 for detailed configurations).

C. PEF details

C.1. Layer-wise perturbation metrics

To evaluate how corruption-induced perturbations propagate through a network, we track the representations of paired clean and corrupted images. For a clean image $\mathbf{x}$ and its corrupted counterpart $\tilde{\mathbf{x}}$, let $A^{(l)}(\mathbf{x})$ denote the output activation tensor at layer l.

Table 2. Shared experimental configuration extracted from the learning-rate search and final training/testing notebooks.

Parameter	**Value**
Global seed	$42, 43$
Dataset	CIFAR-10
Train / validation split	$85\%/15\%$ stratified split
Train images	42,500
Validation images	7,500
Image size	224×224
Resize interpolation	Bicubic
Normalization	ImageNet mean and standard deviation
Batch size	32
Training DataLoader workers	4
Training `pin_memory`	`True`
Final CIFAR-10-C test workers	0
Final CIFAR-10-C test `pin_memory`	`False`
Optimizer	AdamW
Weight decay	10^{-4}
Label smoothing	0.0
Learning-rate scheduler	None
Gradient clipping	None
Active corruption types	3
Active corruptions	`speckle_noise`, `spatter`, `saturate`
Training severity	5
Final test severities	$1, 2, 3, 4, 5$
aPAR discovery images	128 base images
aPAR batch size	8
aPAR epsilon	10^{-8}
Targeted layer count k	4
Adapter hidden ratio	0.25
LoRA rank	8
LoRA alpha	16.0
LoRA dropout	0.0

To ensure representations remain dimensionally comparable across varying network depths—especially after spatial downsampling or pooling operations—we first standardize the activation tensors. For convolutional feature maps where $A^{(l)}(\mathbf{x}) \in \mathbb{R}^{C\times H\times W}$, we apply Global Average Pooling (GAP) across the spatial dimensions H and W, yielding a 1D feature vector $\mathbf{v}^{(l)}(\mathbf{x}) \in \mathbb{R}^{C}$.

Relative Representation Drift (RRD). We measure the total accumulated error at a given layer using the Relative Representation Drift (RRD), defined as the L_2-normalized distance between the clean and corrupted feature vectors:

$$d^{(l)}(\mathbf{x}, \tilde{\mathbf{x}}) = \frac{\|\mathbf{v}^{(l)}(\mathbf{x}) - \mathbf{v}^{(l)}(\tilde{\mathbf{x}})\|_2}{\|\mathbf{v}^{(l)}(\mathbf{x})\|_2 + \epsilon}. \tag{23}$$

The denominator acts as a scale-invariant normalizer, expressing the perturbation size relative to the magnitude of the clean representation. To prevent division by zero in zero-activation regions, we enforce a numerical stability constant of $\epsilon = 10^{-8}$ across all experiments. This quantity represents the cumulative perturbation drift that has survived up to layer l.

Perturbation Amplification Rate (PAR). Because RRD measures cumulative drift, it obscures the marginal contribution of any individual layer. To isolate how layer l locally transforms the incoming perturbation propagating from layer $l-1$, we introduce the Perturbation Amplification Rate (PAR):

$$r^{(l)}(\mathbf{x}, \tilde{\mathbf{x}}) = \frac{d^{(l)}(\mathbf{x}, \tilde{\mathbf{x}})}{d^{(l-1)}(\mathbf{x}, \tilde{\mathbf{x}}) + \epsilon}. \tag{24}$$

PAR acts as an empirical, local multiplicative scaling factor that governs the step-to-step evolution of the error. It partitions the layer's behavior into three distinct transport regimes:

- **Amplification ($r^{(l)}(\mathbf{x}, \tilde{\mathbf{x}}) > 1.0$):** The layer exacerbates the incoming drift, pushing the corrupted representation further from the clean manifold.
- **Suppression ($r^{(l)}(\mathbf{x}, \tilde{\mathbf{x}}) < 1.0$):** The layer filters or attenuates the noise, pulling the corrupted representation closer to the clean manifold.
- **Neutral Propagation ($r^{(l)}(\mathbf{x}, \tilde{\mathbf{x}}) \approx 1.0$):** The layer mostly preserves the relative magnitude of the incoming perturbation.

Adjusted Perturbation Amplification Rate (aPAR). For the purpose of targeted intervention, a layer may represent a critical vulnerability if it exhibits extreme non-neutral behavior in *either* direction—massive amplification can overwhelm downstream layers, while massive suppression may indicate the destruction of task-relevant signal.

To create a stable ranking for intervention, we first compute the expected PAR over a discovery dataset $\mathcal{D}_{\text{disc}}$ (consisting of 128 base images, as detailed in Appendix B.3):

$$\bar{r}^{(l)} = \frac{1}{|\mathcal{D}_{\text{disc}}|} \sum_{(\mathbf{x}, \tilde{\mathbf{x}}) \in \mathcal{D}_{\text{disc}}} r^{(l)}(\mathbf{x}, \tilde{\mathbf{x}}). \tag{25}$$

We then define the direction-agnostic score, the adjusted Perturbation Amplification Rate (aPAR):

$$R^{(l)} = \max\left(\bar{r}^{(l)}, \frac{1}{\bar{r}^{(l)}}\right). \tag{26}$$

This piecewise transformation establishes a symmetric penalty around the neutral baseline of 1.0. For example, a layer that doubles the relative drift ($\bar{r}^{(l)} = 2.0$) and a layer that halves the relative drift ($\bar{r}^{(l)} = 0.5$) will both yield an aPAR of 2.0. As mathematically formalized in Appendix A, aPAR strictly maps to the absolute log-deviation from neutral transport, providing a mathematically grounded heuristic for identifying layers with highly sensitive perturbation dynamics.

C.2. PAR profile computation

For diagnostic amplification-profile figures, we evaluate pretrained ResNet-50 and ConvNeXt-Tiny on matched clean/corrupted CIFAR-10 image pairs across the standard corruption taxonomy. This produces the architecture-level amplification profiles shown in the main text. Specifically, the plotted PAR value for each layer represents the mean PAR ($\bar{r}^{(l)}$) averaged across all images, corruption types, and severities in the full evaluation dataset (for the detailed PAR profiles, refer to Figure 5 and Figure 6).

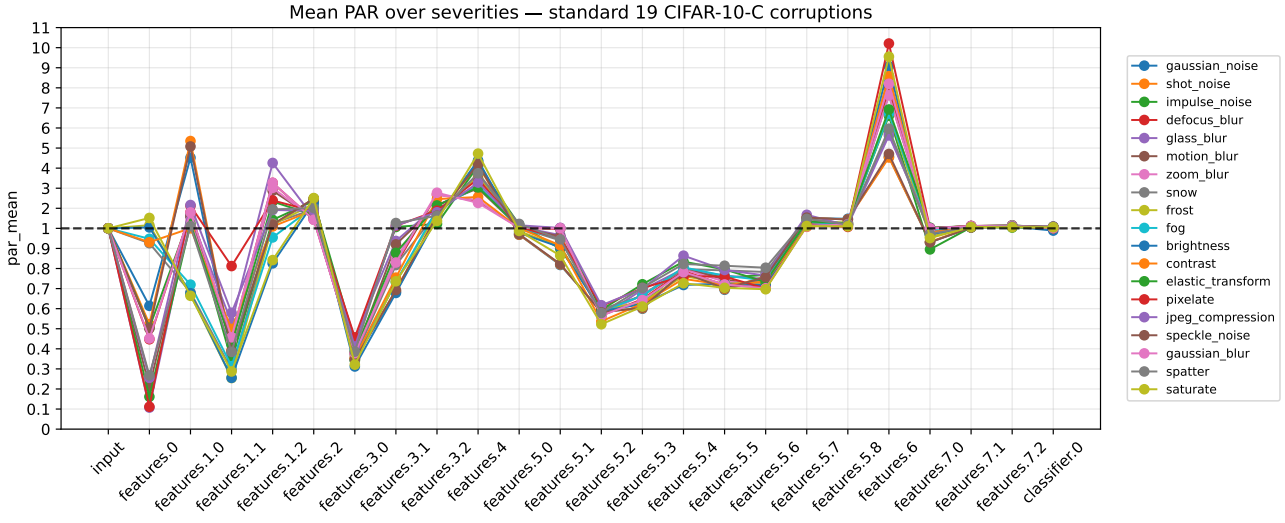


Figure 5. PAR profile for ConvNeXt-Tiny evaluated across 19 standard CIFAR-10-C corruptions. The raw PAR metric captures the architecture's variable layer-wise response to input noise, highlighting oscillations between active perturbation suppression (PAR < 1.0) and amplification (PAR > 1.0).

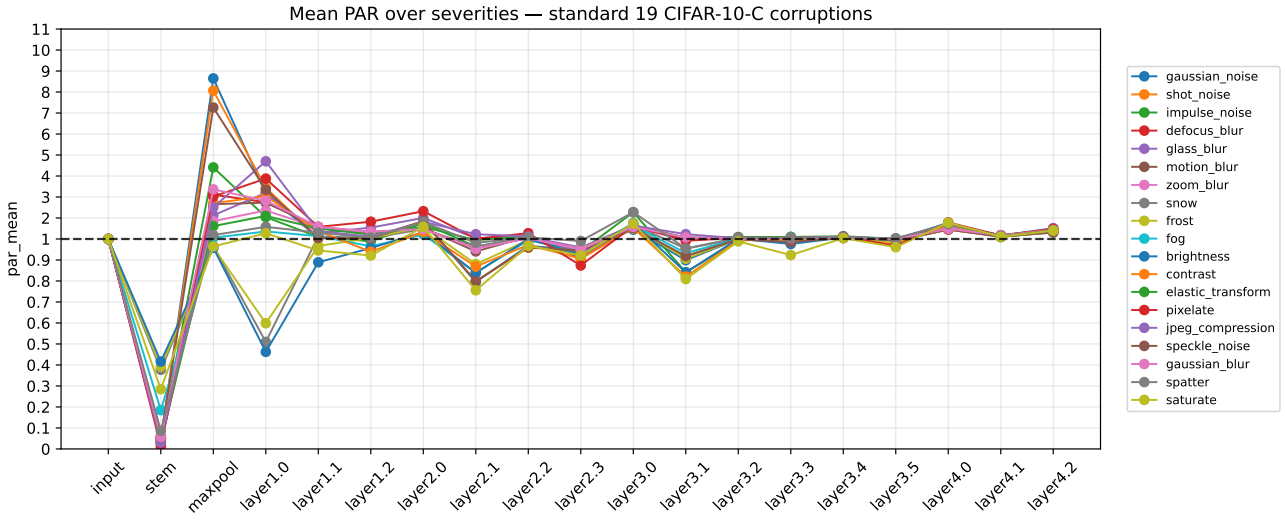


Figure 6. PAR profile for ResNet-50 evaluated across 19 standard CIFAR-10-C corruptions. In contrast to ConvNeXt-Tiny, the raw PAR metric reveals that ResNet-50's structural vulnerabilities are mostly localized in its early layers. It exhibits perturbation suppression at the `stem` followed immediately by an amplification at `maxpool`, before stabilizing into a neutral transport regime (PAR ≈ 1.0) for the remainder of the network.

C.3. aPAR profile computation

The diagnostic aPAR profiles are formulated by applying the symmetric aPAR transformation (Equation 26) on the dataset-wide mean PAR ($\bar{r}^{(l)}$) (Equation 25) to fold the extreme suppression and amplification regimes together (Figure 7 and Figure 8).

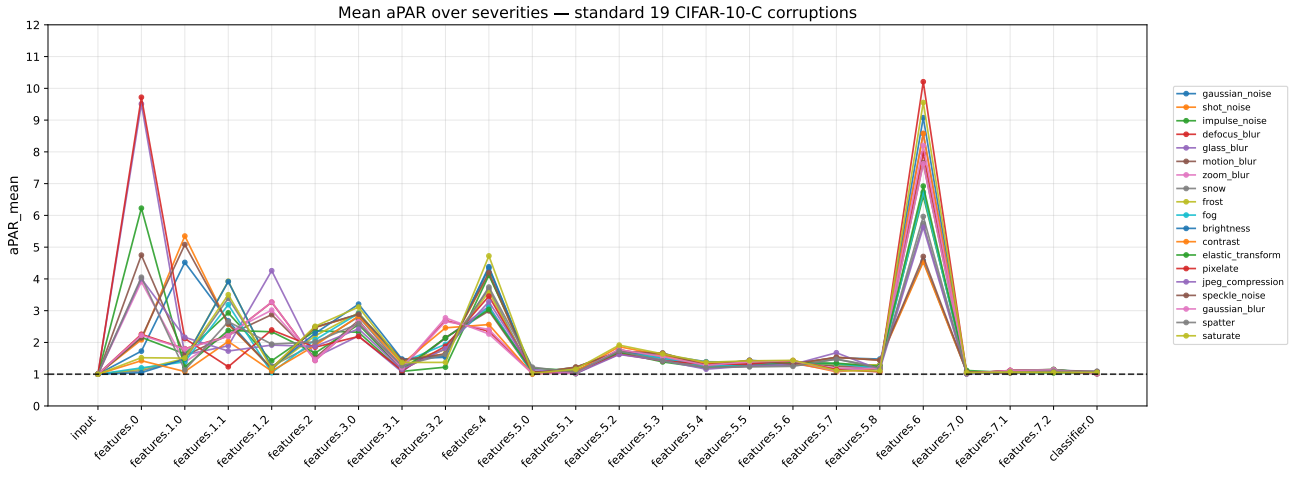


Figure 7. aPAR profile for ConvNeXt-Tiny. The aPAR transformation highlights layers with large deviations from neutral propagation, providing candidate regions for sensitivity analysis and targeted stabilization.

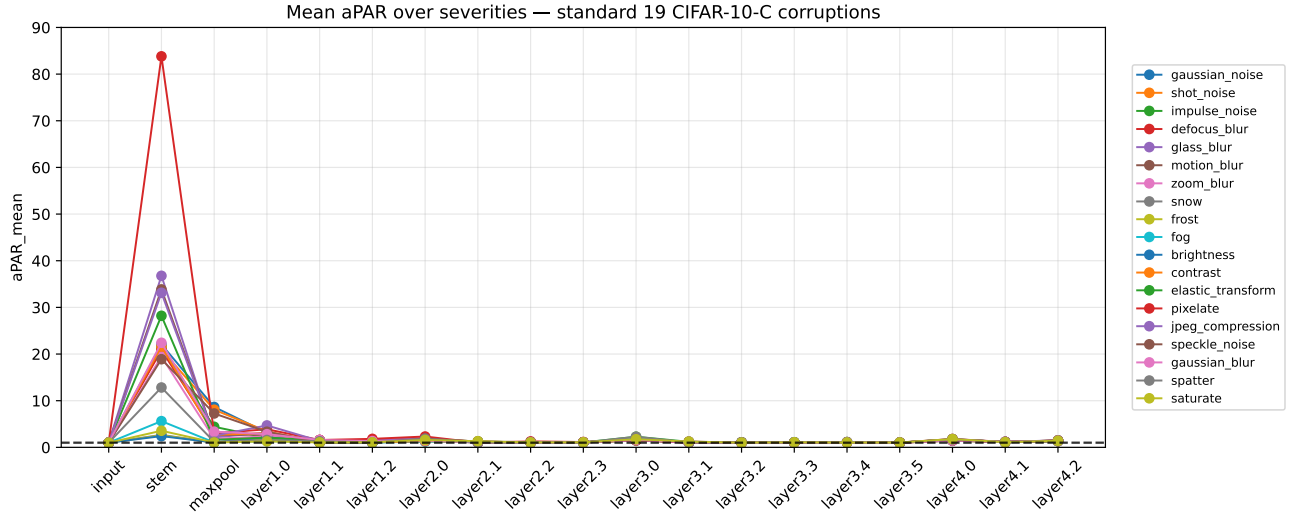


Figure 8. aPAR profile for ResNet-50. For this model, aPAR assigns the largest scores to early layers, consistent with the observed front-loaded perturbation profile, and showing that these layers are potential targets for intervention.

D. Targeted intervention protocol

D.1. Intervention methods

We compare intervention strategies that differ only in which parameters are trainable. The backbone architecture, data split, active corruption subset, corruption severity, optimizer, and learning-rate protocol are otherwise held fixed.

Pretrained reference. The pretrained reference evaluates the ImageNet-pretrained model without CIFAR-10 training. Because the original ImageNet head outputs 1000 classes, while CIFAR-10 and CIFAR-10-C only have 10 classes, the prediction cannot be used directly for CIFAR-10 accuracy. Instead, we group related ImageNet classes into each CIFAR-10 class and sum their probabilities to produce a 10-class prediction. This keeps the pretrained model unchanged and measures how much useful ImageNet knowledge transfers to CIFAR-10 without training.

Full fine-tuning. Full fine-tuning updates all backbone parameters and the classifier head (Yosinski et al., 2014). For ResNet-50, this corresponds to 23.53M trainable parameters in the uploaded implementation. This is the most expensive

adaptation baseline because the full pretrained model is allowed to adapt to the corrupted training distribution.

Partial fine-tuning. Partial fine-tuning updates only the native parameters of the selected target layers and the classifier head. All other backbone parameters remain frozen. No additional modules are inserted. We evaluate top-k, bottom-k, and random-k layer selections.

Adapter tuning. Adapter tuning freezes the original backbone and inserts a lightweight residual adapter into each selected layer (Rebuffi et al., 2017). Each adapter has bottleneck ratio 0.25. The adapter output is added residually:

$$\mathbf{h}' = \mathbf{h} + g(\mathbf{h}) \tag{27}$$

where $g(\cdot)$ is a small trainable bottleneck module. In our implementation, adapters are corruption-specific: each selected layer contains one adapter branch per active corruption type, and the branch is chosen using the local corruption identity of the training sample. Only adapter parameters and the classifier head are trainable.

LoRA tuning. LoRA tuning freezes the original backbone and inserts trainable low-rank updates into selected layers (Hu et al., 2022). For a selected weight matrix W, the effective weight is:

$$W' = W + \Delta W, \qquad \Delta W = BA \tag{28}$$

where A and B are trainable low-rank matrices. We use rank 8, alpha 16.0, and dropout 0.0. Only LoRA parameters and the classifier head are trainable.

D.2. Layer selection

For targeted methods, we use $k = 4$ selected layers. To ensure computational efficiency during training, the aPAR scores used for this selection are not computed over the full 19-corruption benchmark. Instead, we compute aPAR on a fixed, lightweight discovery set drawn from the training split. This discovery set contains 128 clean base images, each paired exclusively with the active training corruption conditions for that run (e.g., `speckle_noise`, `spatter`, and `saturate` at severity 5).

Layer selection is performed exactly once on this discovery set before learning-rate tuning, and is not repeated during hyperparameter search. This prevents layer selection from being confounded with learning-rate selection.

We evaluate three selection rules:

- **Top-k aPAR:** select the k layers with the largest adjusted PAR $R^{(l)}$.
- **Bottom-k aPAR:** select the k layers with smallest adjusted PAR $R^{(l)}$.
- **Random-k:** sample k targetable layers uniformly at random using the active global seed.

For fairness, once a top-k, bottom-k, or random-k layer set is chosen, the same layer set is reused across all intervention methods corresponding to that selection rule. For example, top-k adapter tuning, top-k LoRA tuning, and top-k partial fine-tuning all use the same selected layer set. Because the active global seed determines the active corruption subset, the evaluated noise distributions and resulting aPAR profiles naturally vary between runs (e.g., seeds 42 vs. 43). Consequently, the specific layers selected by the top-k and bottom-k rules also differ by seed. Since parameter counts vary across different network modules, these seed-dependent layer selections cause minor fluctuations in the total trainable parameters across runs. The random-k baseline exhibits similar variance due to its random layer sampling.

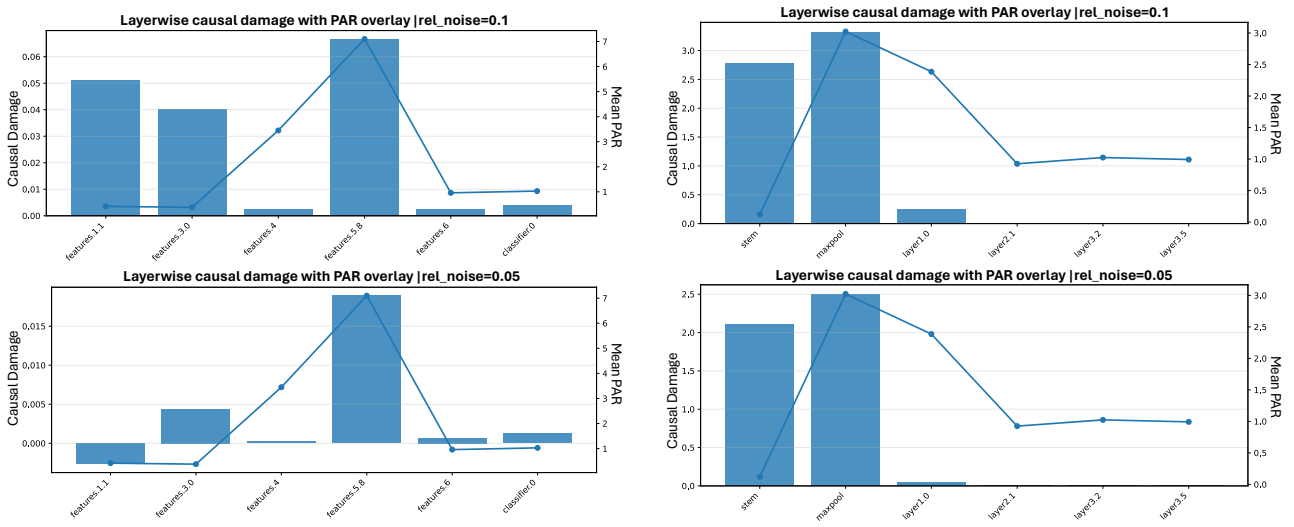
E. Internal causal injection protocol

To test whether PAR/aPAR identifies layers that are causally relevant to downstream robustness, we perform norm-controlled internal noise injection. For a clean input $\mathbf{x}$ and layer activation $h^{(l)}$, we replace the activation with:

$$\tilde{h}^{(l)} = h^{(l)} + \nu^{(l)} \tag{29}$$

where $\nu^{(l)}$ is Gaussian noise scaled relative to the activation norm. The remaining layers of the network are evaluated without modification. This isolates the effect of perturbing one internal location while controlling for differences in activation scale across layers.

We measure causal damage as the degradation in downstream classification performance after injection. Layers are grouped by their PAR/aPAR regime, and the resulting damage is compared across high-, middle-, and low-regime groups (Figure 9). This experiment tests whether extreme perturbation-transport behavior corresponds to layers whose disruption produces greater downstream failure.



(a) ConvNeXt-Tiny

(b) ResNet-50

Figure 9. Layer-wise causal damage (bars, left axis) overlaid with Mean PAR profiles (lines, right axis) for (a) ConvNeXt-Tiny and (b) ResNet-50 under two noise injection severities. The visualization suggests an association between perturbation transport anomalies and causal vulnerability: downstream accuracy degrades most severely when noise is injected into layers exhibiting extreme PAR values—either high amplification (e.g., `maxpool`, `features.5.8`) or high suppression (e.g., `stem`, `features.1.1`). Conversely, layers maintaining neutral transport (PAR ≈ 1.0) induce minimal causal damage.

F. Main experiment

F.1. Two-stage learning-rate search

Learning rates are tuned separately for each architecture and intervention method. We use a two-stage procedure: a coarse Stage-1 search to identify a stable learning-rate region, followed by a Stage-2 refinement inside that region. The learning-rate values themselves are method-specific outputs of this procedure and are not shared across training cases (refer to Table 3 for detailed configurations).

Stage 1: Coarse search. Stage 1 samples learning rates log-uniformly over the broad interval $[10^{-6}, 10^{-2}]$. The search uses 10 learning-rate trials, one random seed, and short training runs. With three active corruption conditions and one corruption-cycle repeat, this corresponds to 3 epochs per trial. Each trial is evaluated using validation accuracy and validation loss on the fixed active corruption subset at severity 5.

Stage-1-to-Stage-2 picker. The LR picker groups runs only by exact learning-rate value; it does not round, bin, or use a weighted score function. Let $M(\eta)$ be the final validation corruption accuracy at learning rate η, expressed in percentage points, and let $L(\eta)$ be the corresponding final validation corruption loss. Let $\eta^\star$ be the learning rate with the best validation

corruption accuracy and let $L^\star = L(\eta^\star)$. A learning rate is considered a stable candidate if it satisfies:

$$M(\eta) \geq M(\eta^\star) - 1.5, \tag{30}$$

$$\frac{L(\eta) - L^\star}{L^\star} \leq 0.35, \tag{31}$$

$$\frac{T_{\text{final}}(\eta)}{T_{\text{initial}}(\eta)} \leq 1.05 \tag{32}$$

where T_{initial} and T_{final} are the first- and final-epoch training losses. All required metrics must also be finite. The picker then identifies contiguous stable regions in the sorted learning-rate list. It prefers a stable region containing the best validation learning rate. If multiple candidate regions remain, it uses deterministic tie-breaking: higher mean validation accuracy, wider region, lower mean validation loss, and lower central learning rate. If no stable plateau is found, the picker falls back to the best finite learning rate and marks the region as weak.

Stage 2: Refined search. Stage 2 constructs 5 candidate learning rates from the selected stable region using a logarithmically spaced grid between the region bounds. It then reuses the same deterministic training and validation loop as stage 1, but with the narrowed Stage-2 learning-rate candidates. In the uploaded ResNet-50 Stage-2 LR-search notebook, the active corruption set contains three corruption conditions and the corruption-cycle repeat is 4, giving 12 epochs per Stage-2 trial. The best Stage-2 learning rate is selected by highest final validation accuracy on the active severity-5 corruption subset.

Table 3. Two-stage learning-rate selection and final training/testing protocol.

Component	**Setting**	**Value**
Stage 1	LR range	$[10^{-6}, 10^{-2}]$
Stage 1	Sampling	Log-uniform
Stage 1	Number of trials	10
Stage 1	Epochs per trial	3
Stage-1-to-Stage-2 picker	Accuracy tolerance	1.5 percentage points
Stage-1-to-Stage-2 picker	Relative validation-loss tolerance	0.35
Stage-1-to-Stage-2 picker	Training-loss ratio tolerance	0.05
Stage-1-to-Stage-2 picker	Minimum stable-region size	2 LR points
Stage 2	Candidate generation	Log-spaced grid inside stable region
Stage 2	Number of candidates	5
Stage 2	Epochs per trial	12 in the uploaded ResNet-50 Stage-2 run
Final training	Selected LR	Best Stage-2 LR for the method
Final training	Epochs	20
Final model selection	Checkpoint criterion	Best validation severity-5 mean accuracy
Final test	Dataset	`randall-lab/cifar10-c`
Final test	Corruptions	Fixed active corruption subset
Final test	Severities	1–5

F.2. Final training

After Stage-2 learning-rate selection, each method is trained for 20 epochs using its selected learning rate. Every run starts from the same ImageNet-pretrained backbone initialization and a newly initialized CIFAR-10 classifier head. The optimizer is AdamW with weight decay 10^{-4} and no learning-rate scheduler. At the end of each epoch, we evaluate both clean validation accuracy and mean validation accuracy over the active severity-5 corruption subset. The checkpoint with the highest validation severity-5 mean accuracy is saved and used for final CIFAR-10-C testing. For reproducibility, each run stores the selected learning rate, trainable parameter count, epoch-level training history, clean validation metrics, corruption validation metrics, and final per-corruption/per-severity CIFAR-10-C subset results.

F.3. Final CIFAR-10-C subset testing

The final test loads the pre-generated CIFAR-10-C data from `randall-lab/cifar10-c`. For each active corruption type and each severity level $s \in \{1, 2, 3, 4, 5\}$, we compute accuracy and cross-entropy loss. The active corruption set is the same fixed subset used during training and validation:

$$\mathcal{C}_{\text{active}} = \{\texttt{speckle\_noise}, \texttt{spatter}, \texttt{saturate}\}.$$

We report the aggregate accuracy over this corruption subset and all five severities:

$$\text{Acc}_{\text{subset}} = \frac{1}{|\mathcal{C}_{\text{active}}| \cdot 5} \sum_{c \in \mathcal{C}_{\text{active}}} \sum_{s=1}^{5} \text{Acc}(c, s).$$

We also record the severity-5 aggregate:

$$\text{Acc}_{s=5} = \frac{1}{|\mathcal{C}_{\text{active}}|} \sum_{c \in \mathcal{C}_{\text{active}}} \text{Acc}(c, 5).$$

The reported CIFAR-10-C result in the targeted stabilization experiments is therefore a fixed-subset CIFAR-10-C accuracy, not the full 19-corruption CIFAR-10-C benchmark accuracy. In the main paper, we denote this metric as "CIFAR-10-C subset accuracy" to avoid conflating the subset evaluation with the full benchmark.

Table 4. Targeted stabilization (using seed 42) results comparing baseline models, full fine-tuning, and parameter-efficient methods (Adapters, LoRA) applied to different layer subsets (top-k, bottom-k, random-k). Results show Clean and CIFAR-10-C accuracy on the subset of 3 corruption types, along with trainable parameter counts for ConvNeXt-Tiny and ResNet-50.

Method	**ConvNeXt-Tiny**			**ResNet-50**		
	Clean acc. ↑	**CIFAR-10-C subset acc. ↑**	**Parameters (M) ↓**	**Clean acc. ↑**	**CIFAR-10-C subset acc. ↑**	**Parameters (M) ↓**
Pretrained	68.29%	49.78%	0.00	54.64%	36.18%	0.00
Full fine-tune	97.29%	95.82%	27.83	95.34%	93.26%	23.53
Adapter (top-k)	91.24%	94.08%	1.18	72.58%	91.43%	8.38
Adapter (bottom-k)	91.42%	93.51%	2.01	71.44%	90.95%	3.95
Adapter (random-k)	90.94%	94.33%	1.56	75.32%	90.02%	5.13
LoRA (top-k)	96.50%	94.29%	0.04	95.20%	92.62%	0.20
LoRA (bottom-k)	96.12%	93.38%	0.81	94.37%	91.33%	0.14
LoRA (random-k)	96.54%	93.98%	0.34	93.47%	90.43%	0.16

Table 5. Targeted stabilization (using seed 43) results comparing baseline models, full fine-tuning, and parameter-efficient methods (adapters, LoRA) applied to different layer subsets (top-k, bottom-k, random-k). Results show Clean and CIFAR-10-C accuracy on the subset of 3 corruption types, along with trainable parameter counts for ConvNeXt-Tiny and ResNet-50.

Method	**ConvNeXt-Tiny**			**ResNet-50**		
	Clean acc. ↑	**CIFAR-10-C subset acc. ↑**	**Parameters (M) ↓**	**Clean acc. ↑**	**CIFAR-10-C subset acc. ↑**	**Parameters (M) ↓**
Pretrained	68.29%	49.36%	0.00	54.64%	34.14%	0.00
Full fine-tune	97.87%	94.09%	27.83	92.20%	89.80%	23.53
Adapter (top-k)	90.53%	86.67%	1.18	68.72%	80.61%	8.38
Adapter (bottom-k)	88.75%	90.16%	2.05	45.25%	80.45%	5.13
Adapter (random-k)	91.72%	86.81%	0.73	67.13%	79.90%	8.67
LoRA (top-k)	93.70%	90.85%	0.04	90.24%	87.24%	0.20
LoRA (bottom-k)	93.37%	90.42%	0.85	88.97%	85.97%	0.16
LoRA (random-k)	92.18%	89.25%	0.40	87.73%	83.80%	0.18

G. Additional experiment

We conduct an additional experiment to verify the model's sensitivity to hyperparameter tuning. In this older learning-rate search pipeline, each architecture and intervention method uses a one-stage search instead of the two-stage procedure used in the main experiments. This experiment is used to examine how strongly the final robustness results depend on the sampled learning-rate candidates.

Learning rates are tuned separately for each architecture and intervention method. For each training case, we sample 5 candidate learning rates log-uniformly over the interval $[10^{-4}, 10^{-2}]$. All candidate learning rates for this ablation are generated using the fixed random seed 42, so the sampled values are reproducible within each notebook.

Each candidate learning rate is evaluated using a short deterministic training run. The model is initialized from ImageNet-pretrained ResNet-50 or ConvNeXt-Tiny weights, and the CIFAR-10 training set is split into 85% training data and 15% validation data using a fixed stratified split. Images are resized to 224×224 and normalized using ImageNet statistics.

The learning-rate search uses a small active corruption subset rather than the full corruption benchmark. Three corruption types are selected using the fixed seed, and only severity level 5 is used during the search. With one corruption-cycle repeat, each learning-rate trial is trained for 3 epochs. Each trial is then evaluated on the clean validation split and on the fixed active corruption subset at severity 5.

For targeted fine-tuning, adapter, and LoRA methods, the selected intervention layers are fixed before the learning-rate search begins. Top-k, bottom-k, and random-k layer selections are therefore kept constant across all learning-rate trials. This ensures that differences between trials come from the learning rate rather than from changing intervention locations.

The best learning rate is selected based on the highest final mean validation accuracy on the active severity-5 corruption subset. This one-stage procedure is lightweight, but it is also sensitive to the sampled candidate set because only 5 learning rates are tested and each trial uses a short 3-epoch run. Unlike the main two-stage procedure, this older pipeline does not identify a stable learning-rate region or perform a refined search within that region. This sensitivity motivates the two-stage learning-rate search used in the main experiments.

For the final evaluation, models are trained following the protocol outlined in Appendix F.2, with the exception that training is extended to 30 epochs.

Table 6. Hyperparameter sensitivity results comparing baseline models, fine-tuning, and parameter-efficient methods (adapters, LoRA) applied to different layer subsets (top-k, bottom-k, random-k). Results show Clean and CIFAR-10-C accuracy on the subset of 3 corruption types, along with trainable parameter counts for ConvNeXt-Tiny and ResNet-50. This experiment shows that the accuracy is sensitive to hyperparameter tuning.

Method	**ConvNeXt-Tiny**			**ResNet-50**	
	Clean acc. ↑	**CIFAR-10-C subset acc.** ↑	**Parameters (M)** ↓	**CIFAR-10-C subset acc.** ↑	**Parameters (M)** ↓
Full fine-tune	96.47%	88.53%	27.83	87.28%	23.53
Finetune (top-k)	97.08%	88.19%	1.56	83.19%	8.03
Finetune (bottom-k)	97.48%	92.58%	11.94	85.04%	3.65
Finetune (random-k)	97.26%	92.22%	3.89	85.27%	3.65
Adapter (top-k)	92.35%	78.38%	1.18	76.27%	8.38
Adapter (bottom-k)	91.39%	69.36%	2.01	77.47%	3.95
Adapter (random-k)	92.10%	54.13%	1.56	76.47%	5.13
LoRA (top-k)	95.95%	87.33%	0.04	78.98%	0.20
LoRA (bottom-k)	95.98%	87.19%	0.81	79.99%	0.14
LoRA (random-k)	95.77%	89.10%	0.34	79.84%	0.16