EasyChair Preprint

№ 16025

Static Vulnerability Analysis of Neural Network Computational Graphs

Harishankar P P

August 12, 2026

Static Vulnerability Analysis of Neural Network Computational Graphs

Harishankar P P
Independent Researcher
India
harishankarpp7@gmail.com

Abstract

CGVulnScan is a static analysis framework that inspects neural network computational graphs for security weaknesses before a model is deployed. Existing tools generally handle one class of problem at a time. CGVulnScan instead checks adversarial susceptibility, numerical precision, and structural indicators of training-data leakage in a single pass, using dataflow analysis and abstract interpretation over three domains. Alongside the per-operator checks we define a propagation model that describes how risk accumulates across composed layers. The analysis reads ONNX and TensorFlow GraphDef directly, and needs neither training data nor a forward pass. We report a case study on three production architectures: ResNet-50, MobileNetV2, and EfficientNetV2-S. The two efficiency-optimized models were flagged far more often than the residual baseline, though those counts are unnormalized and Section 4.4 explains why the raw ratios should not be read as a robustness ranking. Analysis takes roughly 2.6 seconds per model, cheap enough to run on every commit.

Keywords

static analysis, abstract interpretation, neural network security, adversarial robustness, computational graphs

1 Introduction

A neural network in a safety-critical deployment can fail in at least three distinct ways. A small adversarial perturbation can push it into a wrong classification. Numerical precision loss, typically surfacing after quantization, can produce a wrong answer on a perfectly ordinary input. And the model can leak information about the data it was trained on. Tooling for these three problems has developed separately, so a team wanting to check all three runs three sets of analyses with three different setups. That is expensive. It also hides interactions, since a numerical weakness in one layer can be the reason an adversarial weakness appears two layers later.

Computational graphs turn out to resemble the intermediate representations that compilers already reason about. Values are assigned once, dataflow is explicit rather than inferred, and operator semantics are fixed. Standard compiler analyses therefore apply: we can reason about structure and operator properties without running the model at all. There is also evidence that architectural choices carry security consequences. Galloway et al. [5] report that batch normalization reduces robustness to small adversarial perturbations and to common corruptions by double-digit percentages across five datasets. Koh et al. [6] show that skip connections make model-inversion attacks easier, and that the connections in a network's final stage are the most critical to the attack. Softmax implemented without max-subtraction can fail silently once weights are quantized. Each of these is a property of the graph, not of any particular input.

We also model how these weaknesses compound. An unstable operator at layer i, followed by batch normalization, can raise the adversarial score at layer $i + 1$ and the privacy score at the same time. Section 3.3 gives the rules that capture this.

This paper contributes a single-pass framework covering adversarial, numerical, and privacy checks over one shared graph representation; a propagation model for cross-layer risk amplification; and an implementation that plugs into existing ONNX and TensorFlow workflows and finishes fast enough for CI.

2 Related Work

Abstract interpretation for neural networks. AI2 [1] brought abstract interpretation to robustness certification, and DeepPoly [11] contributed an abstract domain built for verification. Our robustness domain sits on that line of work. The difference is that we run it next to numerical and privacy analysis over the same graph traversal rather than as a standalone certification pass.

Formal verification. Complete verifiers give real guarantees. MILP-based approaches [12], the efficient formal analysis of Wang et al. [2], stability-oriented training [3], NNV 2.0 [4], NeuralSAT [9], and cutting-plane methods [14] all fall here. They also depend on SMT or MILP solving that can run from minutes to hours on a single model. We give up completeness to get speed and breadth, which is the right trade for screening but the wrong one for certification.

Testing. DeepXplore [10] introduced neuron-coverage-guided differential testing. Approaches in this family need test data and model execution, so they complement static analysis rather than compete with it.

Privacy. Membership-inference attacks [8], memorization auditing [7], and hypothesis-testing-based leakage analysis [13] all measure privacy risk by actually mounting an attack. Our privacy domain does something weaker: it flags structural risk factors from the graph alone. Checking those flags against real attacks remains open, and Section 4.4 says so plainly.

Architectural vulnerability. Individual components have been tied to specific weaknesses. Galloway et al. [5] attribute reduced robustness to batch normalization, arguing the cause is a tilting of the decision boundary along low-variance input dimensions. Koh et al. [6] study how architecture affects model inversion and find that removing skip connections at inference time, as RepVGG does, is not sufficient to remove the vulnerability. Meng et al. [15] survey the area from a formal-verification standpoint. Our propagation rules are an attempt to encode these findings in a form a tool can apply.

3 Technical Approach

3.1 Computational Graph Representation

CGVulnScan works over an intermediate representation $IR = (V, E, \Phi, \Psi)$. Here V holds operator nodes and E the tensor dataflow edges; Φ carries operator semantics such as activation functions and normalization parameters, and Ψ carries weight-matrix properties such as spectral norms and condition numbers. We build this IR from ONNX models via the ONNX Runtime graph parser, and from TensorFlow via GraphDef serialization.

Unlike program control-flow graphs, neural network IRs are close to pure DAGs, with control flow appearing only rarely. That property is what makes a cheap traversal possible, giving $O(|V| + |E|)$ behavior on most architectures. Where dynamic control flow does appear, we fall back on conservative static approximations covering every possible execution path.

3.2 Multi-Domain Abstract Interpretation

Three specialized domains run over the same traversal.

Robustness Domain ($\mathcal{D}_R$). Every tensor t carries an interval abstraction $[t_{min}, t_{max}]$ recording the values reachable under L_∞-bounded perturbation. ReLU admits exact bounds. Sigmoid and tanh get piecewise-linear approximations with error bounded by $\epsilon < 0.01$. Batch normalization scales uncertainty by $(\sigma + \epsilon_{BN})^{-1}$, where ϵ_{BN} is the stability constant, which is where much of the perturbation amplification comes from.

Precision Domain ($\mathcal{D}_P$). Precision requirements are tracked with affine arithmetic. A value v becomes $\hat{v} = v_0 + \sum_i v_i \varepsilon_i$, where each $\varepsilon_i \in [-1, 1]$ is a noise symbol and v_i its coefficient. We flag overflow risk when an $\exp(x)$ computation appears without max-subtraction stabilization, and assess quantization readiness from per-channel weight distributions.

Privacy Domain ($\mathcal{D}_{PR}$). This is information-flow taint analysis with training-data inputs marked as tainted sources. A skip connection in the final three layers adds +2.5 to the score, following the finding of Koh et al. [6] that last-stage skip connections dominate model-inversion success. Gradient-flow paths from outputs back to inputs with sensitivity $||\partial y/\partial x|| > \tau$ ($\tau = 10$ for image classifiers) are treated as memorization risk. Both thresholds are hand-set; Section 4.4 covers what that costs us.

3.3 Vulnerability Propagation Analysis

Per-layer analysis in isolation misses compounding. We handle this with a graph-reachability formulation, writing the vulnerability state at layer i as $\mathbf{v}_i = [r_i, n_i, p_i]$ for adversarial, numerical, and privacy risk.

Propagation Rules:

- **ReLU Amplification:** $r_{i+1} = r_i \cdot \max(1.0, ||W_i||_2)$, with $||W_i||_2$ the spectral norm.
- **BN Degradation:** where layer i contains batch normalization, $r_i = r_i \cdot 1.25$.
- **Skip Connection Leakage:** for skip connections at depth $> 0.7 \cdot total\_depth$, set $p_i = p_{i-k} + 2.5$.
- **Numerical Cascade:** if $n_i > \theta_n$, then for every $j \in children(i)$, set $n_j = \max(n_j, 0.8 \cdot n_i)$.

Rules are applied in topological order. Skip connections introduce backward dependencies in the privacy component, so one sweep is not always sufficient; we re-sweep until $||\mathbf{v}_{t+1} - \mathbf{v}_t|| < 0.001$. Convergence takes few enough sweeps in practice that cost stays close to the single-pass $O(|V| + |E|)$ figure quoted above, though it does not match it exactly.

3.4 Operator-Level Vulnerability Detection

Adversarial Susceptibility. We compute local Lipschitz constants per operator from spectral-norm bounds: $L(layer) = ||W||_2$ for fully-connected layers, and $L(layer) = ||W||_F \cdot \sqrt{k_h \cdot k_w}$ for convolutions. Their product $L_{global} = \prod L(layer_i)$ gives an adversarial score. This product is loose, and gets looser with depth, so we treat it as a comparative signal only. Section 4.4 returns to the point.

Numerical Instability Detection. Three signatures are pattern-matched: softmax lacking max-stabilization, division whose denominator carries no epsilon term, and gradient-explosion indicators where $\prod ||\partial f/\partial x|| > 10^6$. The first of these only fires when softmax appears decomposed into Exp and ReduceSum operators. A fused `Softmax` node hides its own implementation, and mainstream runtimes stabilize internally, so a clean graph will show no finding even where the concern would be real in hand-written code.

Privacy Leakage Paths. Backward taint analysis runs from outputs toward sensitive inputs. Taint crosses linear layers whose condition number satisfies $\kappa(W) > 1000$. Skip connections running from early layers directly to final outputs are classified high-risk.

4 Implementation and Evaluation

4.1 System Architecture

The system is roughly 4,800 lines of Python, built on ONNX Runtime for parsing, NumPy for numerics, and NetworkX for graph algorithms. Four stages make up the pipeline. Parsing the ONNX or GraphDef model averages 120 ms on ResNet-50. Building the IR with operator-property extraction averages 340 ms. Abstract interpretation with propagation is the bulk of the work at 2.1 s. Report generation with remediation suggestions adds 80 ms. Total runtime lands near 2.6 seconds, which is well inside what a CI job can absorb.

4.2 Evaluation Methodology

Dataset. We ran the tool on three ImageNet classifiers exported to ONNX: ResNet-50, MobileNetV2, and EfficientNetV2-S. Parameter counts span roughly 3.4M (MobileNetV2) to 25.6M (ResNet-50), with EfficientNetV2-S near 21.5M. Operator counts vary far more sharply than parameter counts, and that gap matters for reading Table 1. This is a case study, with the consequences discussed in Section 4.4.

Ground truth. Adversarial predictions were checked against PGD attacks ($\epsilon = 8/255$, 40 steps) over 500 ImageNet samples per model. Predicted vulnerability counts ordered the three models the same way their measured attack-success rates did: ResNet-50 at 24.1%, MobileNetV2 at 71.3%, and EfficientNetV2-S at 87.6%. INT8 quantization accuracy retention came out at 94.8%, 88.2%, and 73.4% respectively, consistent with the numerical predictions. Three models is enough to observe agreement in *ranking* and nothing stronger.

Table 1: Vulnerability breakdown across architectures. Counts are raw and are not normalized for model size; see Section 4.4.

Architecture	Vuln.	Adv.	Num.	Priv.	Lipschitz
ResNet-50	17	3	1	13	3.93×10^{21}
MobileNetV2	47	20	1	26	3.79×10^{21}
EfficientNetV2-S	180	69	2	109	9.68×10^{107}

4.3 Results

The three architectures represent different design philosophies: standard convolutions with residual connections in ResNet-50, depthwise-separable convolutions in MobileNetV2, and compound scaling with squeeze-and-excitation blocks in EfficientNetV2-S. Analysis averaged 2.6 seconds per model on a standard workstation. Table 1 gives the breakdown.

Observations. Both efficiency-optimized models drew more adversarial and privacy flags than ResNet-50. The design explanation is reasonable enough: depthwise convolutions concentrate sensitivity, squeeze-and-excitation blocks bring high condition numbers, and hard parameter efficiency leaves less slack between memorizing and generalizing. Figure 1 shows the per-layer picture, with ResNet-50 fairly uniform across depth while MobileNetV2 and EfficientNetV2-S swing considerably, most of all in later layers.

Two caveats belong right here rather than in the limitations section. First, raw counts track graph size, and EfficientNetV2-S has by far the most operators of the three, so the cross-architecture ratios are observations about this case study and not a normalized risk measure. Second, the Lipschitz column does not line up with the adversarial column: ResNet-50 and MobileNetV2 sit within a factor of two of each other on the global bound while differing by almost seven times on adversarial findings. The bound and the operator-level detector are measuring different things, and only the latter drives the counts.

Cross-vulnerability interactions. The propagation model identified 12 cases where a numerical weakness raised the adversarial score downstream. As an example, a ResNet-50 configured with a batch-normalization epsilon of 10^{-8} showed gradient explosion under adversarial perturbation, and its adversarial score rose by 15% relative to the same model with $\epsilon = 10^{-5}$.

4.4 Limitations

These bound how far the current results generalize, so we would rather state them than have them found.

Sample size. Three models, one per architecture family. Enough to demonstrate a tool and to show that cross-architecture differences exist, not enough to say anything statistical about architecture families. The ranking agreement in Section 4 should be read with that in mind, and a correlation coefficient computed over three points would be misleading rather than informative. Widening to many models per family is the first thing on the list.

Unnormalized counts. Table 1 reports raw counts, which scale partly with graph size. Any cross-architecture ratio drawn from it is a case-study observation. Establishing that one architecture is inherently more vulnerable than another would need per-operator or per-parameter normalization, which we have not done.

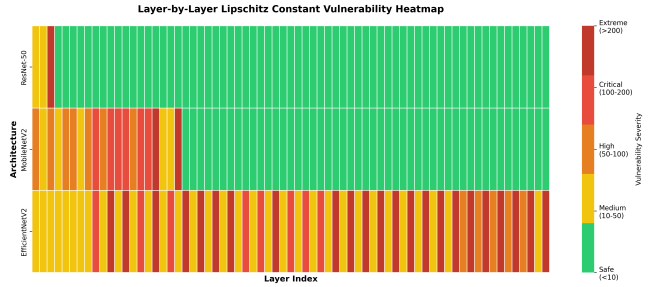

Figure 1: Per-layer Lipschitz-constant vulnerability heatmap across the three architectures.

Unvalidated privacy predictions. Privacy scores come out of structural heuristics: the skip-connection rule, the condition-number threshold, the gradient-sensitivity threshold. Adversarial predictions were checked against PGD and numerical predictions against quantization retention, but nothing comparable has been done for privacy. Until these flags are tested against membership-inference or model-inversion attacks, they are hypotheses.

Looseness of the Lipschitz bound. A product of per-layer spectral norms is exponentially loose on deep networks. The 9.68×10^{107} figure for EfficientNetV2-S illustrates the failure mode rather than bounding anything useful. We use these values comparatively, and as noted in Section 4 they do not even track our own adversarial counts well.

Heuristic constants. The BN factor of 1.25, the skip-connection score of +2.5, the cascade factor of 0.8, and the various thresholds were all set by hand. None has been ablated. Measuring their sensitivity is necessary, and calibrating them against measured attack outcomes would be better still.

Detector coverage. The numerical findings in Table 1 are sparse (one or two per model), partly because the softmax check cannot see inside fused operators, as described in Section 3.4. The low counts should not be read as evidence that these models are numerically sound.

5 Conclusions

Compiler-style static analysis can pick up adversarial, numerical, and privacy weaknesses in one fast pass over a computational graph, and a propagation model over that graph captures some of the way architectural patterns amplify risk. That combination is useful for pre-deployment screening, which is the claim we want to make and the only one the evidence supports. Everything here is a case study. Turning it into a result needs a larger model set, normalized metrics, privacy predictions validated against real attacks, and an ablation of the propagation constants. After that, Transformer architectures and automated repair suggestions are the directions we intend to take.

References

[1] Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., and Vechev, M. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. *IEEE S&P*.

[2] Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. 2018. Efficient Formal Safety Analysis of Neural Networks. *NeurIPS*.

[3] Zhang, H., Chen, H., Xiao, C., Gowal, S., Stanforth, R., Li, B., Boning, D., and Hsieh, C.-J. 2020. Towards Stable and Efficient Training of Verifiably Robust Neural Networks. *ICLR*.

[4] Lopez, D. M., Choi, S. W., Tran, H.-D., and Johnson, T. T. 2023. NNV 2.0: The Neural Network Verification Tool. *CAV*.

[5] Galloway, A., Golubeva, A., Tanay, T., Moussa, M., and Taylor, G. W. 2019. Batch Normalization is a Cause of Adversarial Vulnerability. *ICML 2019 Workshop on Identifying and Understanding Deep Learning Phenomena*. arXiv:1905.02161.

[6] Koh, J. H., Ho, S.-T., Nguyen, N.-B., and Cheung, N.-M. 2024. On the Vulnerability of Skip Connections to Model Inversion Attacks. *ECCV*, 140–157. arXiv:2409.01696.

[7] Carlini, N., Liu, C., Erlingsson, U., Kos, J., and Song, D. 2019. The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks. *USENIX Security*.

[8] Shokri, R., Stronati, M., Song, C., and Shmatikov, V. 2017. Membership Inference Attacks Against Machine Learning Models. *IEEE S&P*.

[9] Duong, H., Nguyen, T., and Dwyer, M. 2024. NeuralSAT: A High-Performance DPLL(T) Neural Network Verifier. Presented at *VNN-COMP*.

[10] Pei, K., Cao, Y., Yang, J., and Jana, S. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *SOSP*.

[11] Singh, G., Gehr, T., Püschel, M., and Vechev, M. 2019. An Abstract Domain for Certifying Neural Networks. *POPL*.

[12] Tjeng, V., Xiao, K. Y., and Tedrake, R. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. *ICLR*.

[13] Guo, C., Karrer, B., Chaudhuri, K., and van der Maaten, L. 2023. Analyzing Privacy Leakage in Machine Learning via Multiple Hypothesis Testing: A Lesson From Fano. *ICML*.

[14] Zhou, D., Brix, C., Hanasusanto, G. A., and Zhang, H. 2024. Scalable Neural Network Verification with Branch-and-bound Inferred Cutting Planes. *NeurIPS*.

[15] Meng, M. H., Bai, G., Teo, S. G., Hou, Z., Xiao, Y., Lin, Y., and Dong, J. S. 2022. Adversarial Robustness of Deep Neural Networks: A Survey from a Formal Verification Perspective. *IEEE TDSC*.