Pareto DNN Verification: Fast for Most Queries

Yizhak Y. Elboher*
The Hebrew University of Jerusalem
Masaryk University
yizhak.elboher@mail.huji.ac.il

Avraham Raviv*
Bar Ilan University
avraham.raviv@biu.ac.il

Amihay Elboher
Ben Gurion University
amihay@bgu.ac.il

Zhouxing Shi
University of California, Riverside
zhouxing.shi@ucr.edu

Hillel Kugler
Bar Ilan University
hillelk@biu.ac.il

Guy Katz
The Hebrew University of Jerusalem
g.katz@mail.huji.ac.il

Abstract

Formal verification of deep neural networks (DNNs) faces a fundamental scalability challenge: verifying robustness is NP-hard and state-of-the-art tools often time out on modern architectures. We propose *PAreto VErification*, or *PAVE*: by augmenting DNNs with early exits (EEs) and verifying exit-by-exit, we solve a large fraction of robustness queries—roughly 80% in our evaluation—in a small fraction of the time required for direct verification. The key insight is that many inputs in a robustness neighborhood exit at an early layer. Verifying this early-exit behavior is often a tractable problem, allowing verification to terminate long before the full network is analyzed. EEs can be added without modifying the base network, and incur at most a 0.3% accuracy change across our tested architectures. We formalize the novel robustness property for EE networks, prove it is Fixed-Parameter Tractable under a trace stability condition (which commonly holds in practice), and present a sound and complete verification algorithm. Two sound optimizations—*break* (early termination when the winner dominates) and *continue* (skipping the inner loop when no runner-up can win)—reduce SAFE-case verification time by up to $10\times$. Experiments on MNIST, CIFAR-10, and CIFAR-100 with networks from 600K to 33M parameters demonstrate that *PAVE* enables solving queries that were not solved otherwise, including formal verification of ResNet-18 on CIFAR-100.

1 Introduction

The Scalability Crisis in DNN Verification. Deep Neural Networks (DNNs) are increasingly deployed in safety-critical domains—from medical diagnostics [19] to autonomous driving—where formal guarantees of correct behavior are essential. Formal verification [13, 20, 45, 48] provides such guarantees by checking whether a network's output satisfies a specified property for *all* inputs in a given domain, thereby ruling out adversarial perturbations [43] and safety violations [7, 37]. However, the problem is NP-hard in general [20], and even the most advanced tools [48, 53] struggle to scale beyond small networks, often timing out on architectures like ResNet-18 or VGG-16. This scalability bottleneck is one of the most pressing open problems in the field [2, 27].

*Equal contribution

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

The Pareto Verification Insight. A well-known property of DNN inference is that many inputs—particularly easy, well-separated ones—can be classified confidently after only a few layers of computation. Early exit (EE) architectures [44, 49] formalize this intuition by inserting intermediate classifiers at various depths; an input exits early whenever its prediction confidence at an intermediate layer exceeds a threshold, saving computation without sacrificing accuracy. We ask: *can this same depth-shortcutting insight accelerate formal verification?*

We answer this question affirmatively, through the *PAVE* framework: by augmenting DNNs with early exits and verifying exit-by-exit, we resolve a large fraction of robustness queries—roughly 80% in our evaluation—in a fraction of the time, since many inputs in a robustness neighborhood are handled at an early exit. For a given input x and perturbation radius ϵ, if all perturbed inputs $x' \in B_\epsilon^x$ exit at the same layer as x (a condition we call *trace stability*), then verifying robustness reduces to checking only the sub-network up to that exit—a strictly smaller and easier problem. We prove that this reduced problem is Fixed-Parameter Tractable (Thm. 2), and we empirically confirm that trace stability holds in the vast majority of `SAFE` cases (Fig. 4), which are commonly the harder ones.

Adding Early Exits without Hurting Accuracy. *PAVE* requires the target DNN to have EEs. The key concern is whether adding EEs degrades the network's predictive accuracy or robustness. We show it does not [33]: EEs are added as thin, frozen auxiliary classifiers that do not alter the base network's parameters. Trained sequentially on residual examples, they preserve the original model's accuracy to within 0.3% across all tested architectures (Table 1). Thus, standard DNNs can be augmented with EEs and subjected to pareto versification as a transparent acceleration technique, applicable alongside base or certified training.

A Novel Formal Framework for EE Verification. While the *PAVE* paradigm is conceptually intuitive, realizing it requires solving a fundamentally new formal problem. Existing verifiers assume a single fixed output layer; they cannot handle EE networks, whose execution follows conditional branching semantics: the output is produced at the *first* exit where the confidence threshold is met. A naive encoding may produce spurious counterexamples where a runner-up wins at exit e while the true winner already prevailed at an earlier exit $e' < e$.

We address this by formalizing a novel robustness property P_{ee} (Sec. 3.1) that explicitly conditions each exit on the absence of earlier winners. We present a baseline sound and complete verification algorithm (Alg. 1), then observe that it may issue many redundant queries when the answer is `SAFE`. We introduce two independent sound optimizations that together yield up to $10\times$ speedup on `SAFE` cases (Alg. 2): (i) **Break**: if the winner always exceeds the threshold at exit k and no prior counterexample exists, verification terminates immediately; (ii) **Continue**: if the winner always exceeds $1 - T$, no runner-up can win at exit k, so the inner per-class loop is skipped. Both optimizations maintain soundness and completeness (proofs appear in App. C).

Contributions. In this work, we contribute to the formal verification of DNNs by: (i) introducing *PAVE* as a general paradigm for accelerating DNN verification—solving a large fraction of queries in a fraction of the time—and showing how EEs enable it without changing the backbone; (ii) formalizing the EE robustness property P_{ee} and proving it is Fixed-Parameter Tractable under trace stability; (iii) presenting a sound and complete baseline algorithm and two sound, efficiency-improving optimizations; (iv) conducting extensive experiments on multiple benchmarks—MNIST, CIFAR-10, CIFAR-100 across FC, CNN, ResNet-18, and VGG-16 architectures—including queries for which the baseline times out; and (v) analyzing the interplay between exit threshold, exit placement, accuracy, and verifiability to give practitioners design guidance.

2 Preliminaries

2.1 Notations

A DNN is represented as a function $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$, where $n, m \in \mathbb{N}$ are input and output dimensions, respectively. For an input $\mathbf{x} \in \mathbb{R}^n$, $\mathcal{N}(\mathbf{x})$ outputs a vector $\mathbf{y} \in \mathbb{R}^m$. We focus on classification networks, where the predicted class is the index of the highest value in $\mathbf{y}$ (the *winner*); other indices are *runner-ups*. An ϵ-ball around x, denoted B_ϵ^x, is the set $\{x' \in \mathbb{R}^n : \|x' - x\|_p \leq \epsilon\}$, where $\|\cdot\|_p$ is the ℓ_p norm defined by a given parameter $p \in [1, \infty)$. For $k \in \mathbb{N}$, we write $[k] = \{1, \ldots, k\}$.

2.2 Formal Verification of DNNs

The formal verification of a DNN $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$ can be cast into a constraint satisfiability problem, where the goal is to determine whether a property P is *satisfiable* in $\mathcal{N}$. P represents the existence of an input to $\mathcal{N}$ within a specific domain $\mathcal{D}$ whose output satisfies a constraint ϕ:

$$\exists \mathbf{x} \in \mathcal{D} \text{ such that } \phi(\mathbf{x}, \mathcal{N}(\mathbf{x})) \text{ is satisfied.}$$

Typically, ϕ captures undesirable behavior by encoding the negation of a desired property. Thus, if P is satisfiable, $\mathcal{N}$ is said to be `UNSAFE`. If P is not satisfiable, the desired property holds and $\mathcal{N}$ is `SAFE`.

2.3 DNNs with Early Exits

A DNN with early exits is a network augmented with additional decision points, known as *exits*, within its architecture. These exits allow the network to terminate inference early if a condition, typically a confidence threshold, is met, reducing computational cost while maintaining accuracy. Let $\mathcal{N}_{ee} : \mathbb{R}^n \to \mathbb{R}^m$ denote a network with k exits, and let $\mathbf{y}^{(j)}$ represent the neuron values of the j-th exit. Inference at exit j terminates if $f(\mathbf{y}^{(j)}) \geq T_j$, where T_j is a predetermined confidence threshold for the j-th exit, and f is a confidence function. In early works [44], $f(\mathbf{y}^{(j)})$ was computed as the entropy of the j'th exit logits; later, alternative gating mechanisms were proposed, including maximum SoftMax probability [35], confidence accumulation [39], and learned exit conditions [51]. Regardless of the gating function f, the final output $\mathbf{y}$ is determined by the first exit satisfying the threshold; if no threshold is met, the output comes from the last exit. In this work, we use a fully connected layer followed by a *SoftMax* activation as the confidence function f, with the condition $\max(\mathbf{y}^{(j)}) > T_j$. A common threshold, also used in many of our experiments, is $T = 0.9$ [9, 33]. Fig. 8 in App. A depicts a DNN with EEs.

3 The *PAVE* Framework

PAVE decomposes the hard problem of verifying a deep network into a sequence of smaller sub-problems, one per exit, exploiting the fact that for typical inputs the verification terminates early. This section formalizes the approach. We begin in Sec. 3.1 by redefining local robustness to accommodate multiple exits. Then, in Sec. 3.2, we present the base algorithm that mirrors EE conditional inference semantics. In Sec. 3.3, we analyze the complexity of this approach and show that, under trace stability, its cost is significantly reduced. Finally, in Sec. 3.4, we introduce two optimizations that reduce redundant queries while preserving soundness and completeness.

Why Standard Verifiers Cannot Handle EE Networks. Networks with EEs introduce two challenges that break standard verification: (1) In conventional DNNs, the output property is defined on a single output layer; for EE networks, this must be adapted for multiple output layers. (2) Multiple exits introduce conditional branching: if an exit is confident it returns; otherwise computation continues. Naive verification ignores this, producing spurious counterexamples where a runner-up wins at exit e while the winner already won at $e' < e$. *PAVE* resolves both issues by formalizing the robustness query to respect EE semantics and leveraging EEs to shorten verification.

3.1 Revised Robustness Property

For a standard DNN $\mathcal{N}$ (without EEs), the property to negate local robustness around a sample x is:

$$P := \exists x' \in B_\epsilon^x, \exists i \in \mathcal{C} \text{ such that } \mathcal{N}(x')_i > \mathcal{N}(x')_w$$

where $\mathcal{C}$ is the set of possible output labels $\{1, \ldots, m\}$, B_ϵ^x is the ϵ-ball around x, w is the index of the winner class and $\mathcal{N}(x)_j$ is the j'th value in $\mathcal{N}(x)$.

In DNNs with EEs, the inference process enables outputs to be returned from various exits, corresponding to different neurons. This adds ambiguity to the traditional definition: the verification process must condition the validity of the specification on the assumption that a specific exit serves as the actual output layer for the given input. A counterexample to robustness of a network with EEs is one where (i) a "runner-up output" wins at an early or output exit e, and (ii) the desired output does

Algorithm 1 Pareto Verification of DNN Robustness

Input $\mathcal{N}$, x, ϵ **Output** $\mathcal{N}$ is robust in B_ϵ^x, counterexample

1: $w, ee, last$ = $argmax(\mathcal{N}(x))$, indices of layers with EEs, last layer index
2: **for** $k \in ee \cup \{last\}$ **do**
3: **for** $i \in \mathcal{C} \setminus \{w\}$ **do**
4: **if** k $\neq$ last **then**
5: $\mathcal{P} := \exists x' \in B_\epsilon^x : (\mathcal{N}(x')_i^k > T_k) \wedge (\forall e \in ee \cap [k-1] : \mathcal{N}(x')_w^e < T_e)$
6: **else**
7: $\mathcal{P} := \exists x' \in B_\epsilon^x : (\mathcal{N}(x')_w < \mathcal{N}(x')_i) \wedge (\forall e \in ee : \mathcal{N}(x')_w^e < T_e)$
8: **end if**
9: res, cex = Verify($\mathcal{N}, B_\epsilon^x, \mathcal{P}$)
10: **if** res == UNSAFE **then**
11: **return** UNSAFE, cex
12: **end if**
13: **end for**
14: **end for**
15: **return** SAFE

Algorithm 2 Pareto Verification with Break and Continue Optimizations

Input $\mathcal{N}$, x, ϵ_p **Output** $\mathcal{N}$ is robust in B_ϵ^x, counterexample

1: $w, ee, last$ = $argmax(\mathcal{N}(x))$, indices of layers with EEs, last layer index
2: **for** $k \in ee \cup \{last\}$ **do**
3: **if** k $\neq$ last **then**
4: $\mathcal{P} := \exists x' \in B_\epsilon^x : \mathcal{N}(x')_w^k < T$
5: **else**
6: $\mathcal{P} := \exists x' \in B_\epsilon^x, \exists i \in \mathcal{C} \setminus \{w\} : \mathcal{N}(x')_w < \mathcal{N}(x')_i$
7: **end if**
8: res, cex = Verify($\mathcal{N}, B_\epsilon^x, \mathcal{P}$)
9: **if** res == SAFE **then**
10: **return** SAFE
11: **end if**
12: res, cex = Verify($\mathcal{N}, B_\epsilon^x, \forall x' \in B_\epsilon^x : \mathcal{N}_w^k(x') < 1 - T$)
13: **if** k == last $\vee$ res == UNSAFE **then**
14: res, cex = $ExistsPrevCEX(\mathcal{N}, x, \epsilon, k)$
15: **if** res == UNSAFE **then**
16: **return** UNSAFE, cex
17: **end if**
18: **end if**
19: **end for**
20: **return** SAFE

not prevail at any exit preceding e. These conditions are encapsulated in the following property P_{ee}, which defines the negation of the revised local robustness property for a DNN with EEs. The indices of the layers with EEs and the index of the output layer are denoted with ee and $last$, respectively.

$$
\begin{aligned}
P_{ee} := \exists x' \in B_\epsilon^x, i \in \mathcal{C} \setminus \{w\}, e \in ee \cup \{last\} : \\
\big((\mathcal{N}(x')_i^e > T_e \wedge e < last) \ \vee \ (\mathcal{N}(x')_i > \mathcal{N}(x')_w \wedge e = last)\big) \ \wedge \\
\forall j \in ee \cap \{1, \ldots, e-1\} : \mathcal{N}(x')_w^j < T_j
\end{aligned}
$$

Here, P_{ee} asserts that (first line) there exists an input x', a runner-up i, and an exit e such that (second line) the runner-up wins at the early exit (left side) or at the last layer (right side), and (third line) the winner does not prevail at any earlier exit. If P_{ee} is satisfiable, the network is UNSAFE for robustness within B_ϵ^x. Otherwise, $\mathcal{N}$ is SAFE.

3.2 Verification Algorithm

To verify robustness in DNNs with EEs, we propose Alg. 1. The algorithm iterates through all exits (outer loop). For each exit, it examines each runner-up class (inner loop) to determine whether there exists a counterexample where the runner-up wins and the output is produced at the current exit. This is accomplished by a verification query (line 5 or 7) that tries to satisfy: the runner-up wins, and the winner has not already won in any preceding exit. If UNSAFE is returned, the resulting example satisfies P_{ee}, providing a counterexample to robustness. Otherwise, if SAFE is returned for all exits and classes, local robustness is verified. The call to Verify on line 9 launches an underlying verification tool to solve a standard query.

Theorem 1. *If the underlying verifier is sound and complete, Alg. 1 is sound and complete.*

PAVE returns SAFE or UNSAFE only when all required subqueries are proved; if any required subquery is UNKNOWN, *PAVE* returns UNKNOWN. Therefore, *PAVE* remains sound with an underlying incomplete verifier, but may be incomplete itself. This limits applicability to operations supported by the chosen verifier: for example, transcendental activations can be used only when the verifier provides a sound encoding or bound for them.

3.3 Fixed-Parameter Tractable Complexity

Alg. 1 contains two nested loops; while it returns immediately upon finding a counterexample (UNSAFE), it must exhaust the entire loop before concluding SAFE. This motivates further improvements, as we show that in some cases, local robustness in DNNs with early exits can be determined more efficiently. For that purpose, we define the *trace* of an input and its stability as follows.

Definition 3.1. The trace $\tau(x)$ of an input x in DNN $\mathcal{N}$ with EEs is the set of layers x is propagated through. Given an $\epsilon > 0$, $\tau(x)$ is *stable* in B_ϵ^x if $\forall x' \in B_\epsilon^x : \tau(x') = \tau(x)$.

The trace of an input determines which parts of the network must be considered to verify robustness for that input. If all $x' \in B_\epsilon^x$ share the same trace as x, then if Alg. 1 does not return UNSAFE before reaching the exit layer of x, it will eventually return SAFE. Hence, under trace stability, the complexity of solving P_{ee} depends on $|\tau(x)|$—the number of layers in $\tau(x)$—rather than the total number of layers in $\mathcal{N}$. In Sec. 4 (Fig. 4), we show that trace stability often holds in practice.

Definition 3.2 ([10, Def. 1])**.** A problem is *Fixed-Parameter Tractable* (*FPT*) with respect to a parameter p, (denoted *FPT*(p)), if it can be solved in time $f(p) \cdot poly(n)$, where f is a computable function of p, and n is the input size.

Theorem 2. *Given a ReLU network $\mathcal{N}$ with EEs, layer width bound k, input x, and $\epsilon > 0$, if $\tau(x)$ is stable in B_ϵ^x, then solving P_{ee} with $(\mathcal{N}, x, \epsilon)$ using an exact verifier is FPT$(k \cdot |\tau(x)|)$.*

This result is the theoretical foundation for *PAVE*: when trace stability holds, verifying a robustness query scales with the depth of the *exit prefix* rather than the full network. With an incomplete verifier, the same prefix reduction still applies operationally, but the FPT statement is a bound on the exact decision problem rather than a completeness guarantee for that verifier. Under typical EE training [44], the trace of most inputs is significantly shorter than the full network, yielding a substantial reduction in worst-case complexity.

3.4 Two Sound Optimizations

Proving SAFE with Alg. 1 requires calling Verify for every exit and every class. We discuss two improvements to expedite the overall procedure. To maintain readability, we denote lines 3–13 in the algorithm as $ExistsPrevCEX(\mathcal{N}, x, \epsilon, k)$, which rules out adversarial examples in earlier layers.

Continue Optimization (orange lines in Alg. 2). One improvement reduces the number of queries per exit by checking whether the winner's score always exceeds $1 - T$. If so, no runner-up can exceed T (since all class scores sum to 1), allowing the inner loop to be skipped. This check is empirically successful in most cases, bypassing the class-by-class inner loop.

Break Optimization (blue lines in Alg. 2). A second improvement terminates verification early by checking whether the winner's score always exceeds the threshold at exit k. If this holds and no counterexample exists in prior exits, then all inputs advance to exit k where the original winner consistently prevails—verification can *break* and soundly return SAFE without checking later exits.

Theorem 3. *If the underlying verification tool is sound and complete, Alg. 2 is sound and complete.*

Theorem 4. *Given a network $\mathcal{N}$ with EEs and ReLU activations, layer width bound k, input x, and $\epsilon > 0$, if $\tau(x)$ is stable in B_ϵ^x, then Alg. 2 runtime is $\mathcal{O}(2^{k \cdot |\tau(x)|}) \cdot poly($#neurons in $\mathcal{N})$.*

To summarize: we formalized a robustness property for DNNs with EEs and proved it is fixed-parameter tractable (Thm. 2) under trace stability. We presented a sound and complete baseline algorithm (Alg. 1), then introduced break-and-continue heuristics (Alg. 2) and showed they yield a tighter complexity bound (Thm. 4). Proofs of soundness and completeness (Thms. 1 and 3) and of complexity results (Thms. 2 and 4) can be found in App. C. The framework extends naturally to other properties and activation functions (see App. G).

4 Evaluation

We evaluate *PAVE* across a diverse set of networks and datasets to answer four questions: (i) Does *PAVE* work in practice and what are its limits? (ii) Does adding EEs improve verifiability compared to standard (no-EE) verification? (iii) Does trace stability hold empirically, explaining the efficiency gains? (iv) How do EE design choices (threshold, exit placement) affect the accuracy–robustness–verifiability trade-off?

4.1 Benchmarks, Experimental Setup and Model Training

We conducted experiments on MNIST [26] and CIFAR-10/100 [23], with multiple architectures: Fully Connected, LeNet-5 (CNN) [25], ResNet-18 [18], and VGG-16 [41]. Full training details appear in App. D.

We adopt the sequential EE training procedure from [44, 56]: a baseline model is first trained without exits; EEs (each a fully connected layer with SoftMax) are then added and trained sequentially, keeping the base model fixed. This ensures that EEs do not degrade the original model's accuracy.

We place exits at the ends of computational blocks and select thresholds on a validation accuracy-latency Pareto curve; these choices preserve the backbone and provide an early-exit population while exposing the accuracy-latency-verifiability trade-off in Sec. 4.2.

We implemented our framework in PyTorch and ran experiments on an NVIDIA A100 GPU (80 GB HBM2e). As an underlying verification tool, our framework uses (but is not limited to) α-β CROWN [48, 52, 53], a SOTA method for formally verifying DNN robustness properties. α-β CROWN does not currently support nested AND operators. We circumvent this by encoding each query as a collection of smaller queries, one per conjunct, and use the same encoding for the original queries for a fair comparison. More details in App. H.

Tab. 1 summarizes the benchmarks. EE accuracy changes by at most approximately 0.3% from the base accuracy. The benchmarks span 600K to over 33M parameters, demonstrating acceleration on moderately large, standard-trained image classifiers. Our claim is the reduction of verification work for a fixed network and specification: difficulty depends on the architecture, specification, perturbed dimensions, radius, training, timeout, and backend, not parameter count alone. For example, the 138M-parameter VGG-16 VNN-COMP benchmark [3] perturbs only a small subset of ImageNet pixels in some instances, whereas our CIFAR queries perturb all 3,072 pixels.

4.2 Evaluating the Practicality of Pareto Verification

We evaluate *PAVE* (Alg. 2) against the standard Vanilla verifier (no EEs) on the four benchmarks of Tab. 1. We used 100 samples per benchmark per ϵ, with $\epsilon \in \{0.1, 0.05, 0.01, 0.005, 0.001\}$ and a 30-minute per-query timeout. Queries that hit the timeout or triggered an assertion failure in the underlying verifier are counted as *unresolved*; note that unresolved outcomes come from the underlying verifier, not from our algorithm.

We report two complementary views of the data. Fig. 1 is a cactus / survival plot: for each benchmark, the number of queries each method resolves within a given wall-clock budget. Fig. 2 breaks the same data down per ϵ: for each epsilon value, the number of queries each method resolves.

***PAVE* solves more queries, and earlier.** Fig. 1 makes two claims at once: on all three CIFAR-10 benchmarks *PAVE* resolves more queries overall (higher plateau) and resolves each individual query faster on average (curve rises earlier). On VGG-16, *PAVE* resolves more than twice as many queries

Table 1: Characteristics and evaluation metrics of the benchmarks.

Model	Dataset	Size	# Layers	Accuracy	EE Accuracy	Exit Distribution
FC-6	MNIST	1,519,720	6	98.18	98.2	[9772, 152, 46, 30]
CNN (LeNet-5)	CIFAR-10	596,178	5	70.02	69.93	[5924, 2066, 2010]
ResNet-18	CIFAR-10	11,243,102	18	86.43	86.11	[6656, 1899, 1445]
VGG-16	CIFAR-10	33,769,566	13	93.45	93.14	[8268, 1278, 454]

as Vanilla (55 vs. 24)—Vanilla times out on most SAFE queries, while *PAVE* resolves them. On ResNet-18, *PAVE* likewise resolves more queries than Vanilla, and on LeNet-5 it finishes the same set of queries substantially earlier. These are the benchmarks where verification cost actually matters, and they are exactly the regime where *PAVE* delivers.

***PAVE* unlocks the hard mid-range ϵ.** Fig. 2 breaks coverage down by ϵ and clarifies *where* the gains come from. At very small ϵ most queries are trivially SAFE and both methods resolve them; at very large ϵ most are trivially UNSAFE and counterexamples are found quickly by both. The interesting regime is the mid-range, where queries are genuinely hard and the underlying verifier spends its budget. This is where *PAVE* dominates: on LeNet-5 at $\epsilon = 0.005$, *PAVE* resolves 39 queries while Vanilla resolves only 2; on VGG-16 at $\epsilon \in \{0.001, 0.01\}$, *PAVE* resolves 17 and 3 queries while Vanilla resolves none; on ResNet-18 at $\epsilon \in \{0.005, 0.01\}$, *PAVE* resolves 5 and 1 queries while Vanilla again resolves none. Missing Vanilla bars in Fig. 2 correspond exactly to these zero-coverage cases. By sidestepping the full network for inputs that exit early, *PAVE* makes tractable the queries that the underlying verifier simply cannot solve within the timeout.

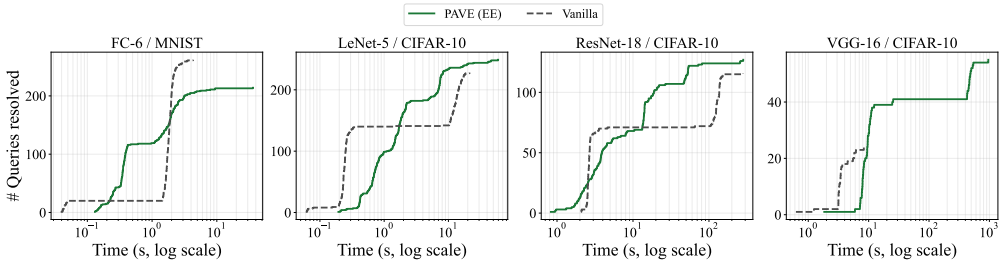


Figure 1: Number of queries resolved (SAFE + UNSAFE) within a given wall-clock budget, per benchmark. Higher and further-left curves are better. *PAVE* (green) rises earlier than Vanilla (gray dashed) on every benchmark and plateaus higher on every CIFAR-10 network. The results show that the *PAVE* benefit grows with network size—modest on the small FC-6 and largest on VGG-16.

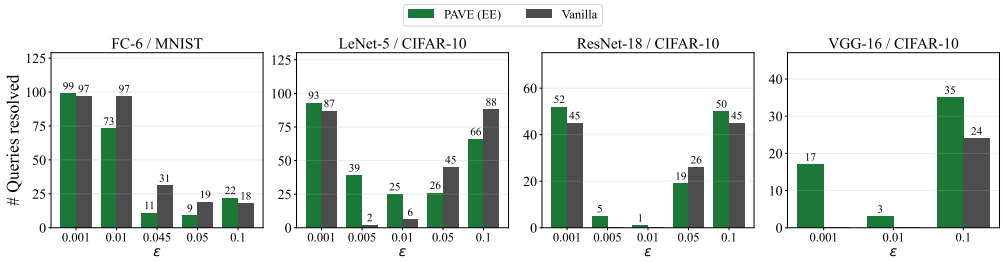


Figure 2: Number of queries resolved per ϵ, on a matched ϵ grid. Missing Vanilla bars indicate that Vanilla resolved *zero* queries at that ϵ within the timeout (ResNet-18 at $\epsilon \in \{0.005, 0.01\}$, VGG-16 at $\epsilon \in \{0.001, 0.01\}$); *PAVE* resolves queries in all of those cells. As in Fig. 1, the benefit grows with network size: on FC-6, Vanilla matches or exceeds *PAVE* coverage; on the larger CIFAR-10 networks, *PAVE* adds coverage in the hard mid-range ϵ.

The *PAVE* advantage grows with network size. Both figures show a clear size-monotone pattern. On the smallest benchmark (FC-6 / MNIST, ≈ 1.5M parameters), Vanilla verification already works well and in fact resolves slightly more queries than *PAVE* overall—this is expected, since FC-6 is small enough that the underlying verifier does not need help with coverage. As networks grow (LeNet-5, ResNet-18, VGG-16), *PAVE* overtakes Vanilla and the gap widens, reaching its maximum on the largest network. This is the intended regime for *PAVE*: the larger and deeper the network, the more the underlying verifier benefits from only having to reason about a shallow prefix. On FC-6, the *PAVE* advantage instead shows up as reduced runtime per query (Tab. 2 and Fig. 3).

Break and continue ablation. Tab. 2 compares Alg. 1 (no optimizations) against Alg. 2 (with the break and continue optimizations). Differences on UNSAFE cases are minor—counterexamples are found quickly regardless—but SAFE cases improve by up to $10\times$. On VGG-16, Alg. 1 could not

Table 2: Verification runtime statistics (seconds) per benchmark. Rows correspond to Alg. 1 (top) and Alg. 2 (bottom).

Benchmark	UNSAFE			SAFE		
	Mean	Std	Median	Mean	Std	Median
MNIST, FC6	0.527 0.600	1.903 2.739	0.068 0.201	13.037 1.143	3.185 1.205	13.032 0.397
CIFAR10, LeNet	2.250 2.279	6.670 4.556	0.320 0.688	56.927 7.255	9.516 10.550	58.628 6.185
CIFAR10, ResNet18	3.518 5.679	12.499 12.316	1.306 3.197	409.395 42.646	74.235 60.521	375.703 19.061
CIFAR10, VGG16	5.412 8.823	1.032 1.818	5.588 8.737	– 1532.418	– 348.150	– 1417.474

prove any SAFE case before timing out, while Alg. 2 succeeded with a 1532-second mean verification time, making VGG-16 verification feasible.

Scaling to CIFAR-100. We additionally verified a ResNet-18 model on CIFAR-100. Both the baseline and the EE model identified UNSAFE cases; however, the EE model verified SAFE examples in about one hour, while the baseline reached a two-hour timeout. At $\epsilon = 0.001$, 14 samples were verified as SAFE only with EEs, while all were UNKNOWN for the baseline—demonstrating that *PAVE* can enable verification of otherwise infeasible cases.

4.3 *PAVE* vs. Classical DNN Verification

We compare verification times for networks with and without EEs to demonstrate the potential of *PAVE* for improving any DNN's verifiability.

Fig. 3 shows that EE-augmented DNNs are verified significantly more efficiently. While verification time for simple queries remains largely unchanged, harder queries benefit from substantial speedups.

To understand *why PAVE* is faster, Fig. 4 compares inference exit layers with verification exit layers. A strong correlation emerges for SAFE cases: inputs that exit at an early layer during inference also verify at that early layer, confirming that trace stability holds in practice. No such correlation appears for UNSAFE cases (counterexamples behave differently from the original sample), which is expected; we discuss why trace stability cannot be expected to hold on UNSAFE inputs in App. I. Additional results for MNIST are in App. E.

4.4 Robustness Analysis and Design Guidelines

Training networks with EEs entails a trade-off between predictive accuracy and inference latency. Different hyperparameter settings yield distinct working points, and users must select the configuration that best satisfies their constraints.

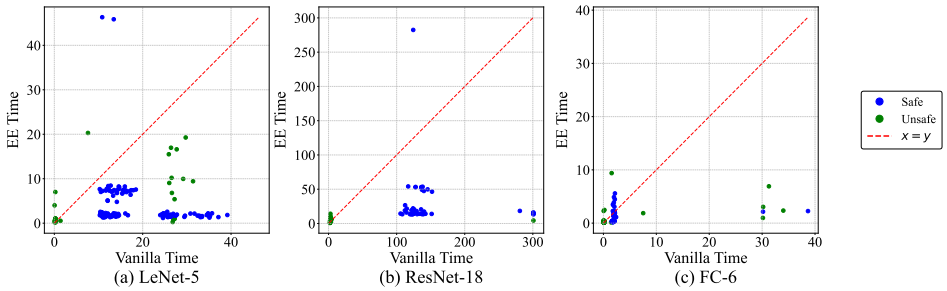
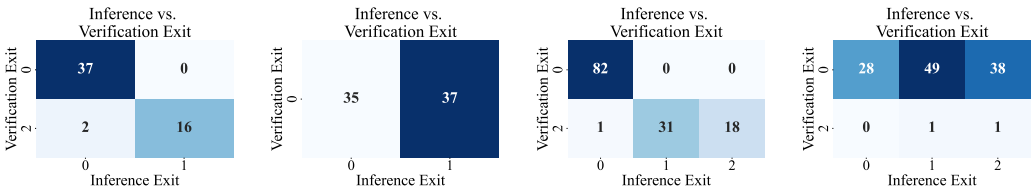


Figure 3: Comparison of verification times between the original model with the underlying verifier (Vanilla) and the model with EEs, using Alg. 2. Each point is a single query; the diagonal marks equal time.



(a) ResNet-18 `SAFE` (b) ResNet-18 `UNSAFE` (c) LeNet-5 `SAFE` (d) LeNet-5 `UNSAFE`

Figure 4: Heatmaps of inference vs. verification exit layer. Strong diagonal correlation for `SAFE` cases confirms that trace stability holds in practice, explaining why *PAVE* terminates early.

Table 3: Verification statistics for ResNet-18 variants with a single early exit.

Architecture	Accuracy	Inference Time	#SAFE	#UNSAFE	#UNKNOWN	Safe Ver Time	Unsafe Ver Time
RN_ee1	0.8757	30.56ms	121	109	190	21.4s	11.7s
RN_ee2	0.8921	36.94ms	85	108	227	23.4s	14.2s

Impact of the Early Stop Threshold. Adjusting the confidence threshold at each exit introduces a trade-off between accuracy and latency. Fig. 5 (left) shows that higher thresholds improve accuracy but increase inference time. To quantify robustness, we measure it as the proportion of inputs formally verified as `SAFE`, i.e. `#SAFE` / (`#SAFE` + `#UNSAFE`). With the EE architecture fixed and only the threshold varied, Fig. 5 (middle) shows that robustness closely follows accuracy—higher thresholds boost both. However, Fig. 5 (right) reveals that verification time also grows with the threshold. Consequently, threshold selection requires balancing accuracy, latency, robustness, and verifiability jointly.

To dissect verification cost further, Fig. 6 breaks down verification times across ϵ values. Both mean and variance grow with the threshold, reinforcing that more conservative exit criteria demand heavier verification effort. Fig. 7 compares against the vanilla network (no EEs, $T = 1$): the vanilla model achieves the highest robustness but requires the longest verification time.

Impact of Exit Location. Fixing the threshold and varying the exit position, we compare two ResNet-18 variants: one with an exit after block 1 (**RN_ee1**) and one after block 2 (**RN_ee2**). Tab. 3 shows that earlier exits increase robustness and verifiability (fewer `UNKNOWN`, shorter verification time) at a minor accuracy cost; later exits preserve accuracy but reduce verifiability.

We find that earlier exits raise robustness and reduce verification time, while later exits trade verifiability for accuracy. These insights give practitioners actionable design guidance for choosing exit locations that best balance their accuracy, latency, robustness, and safety requirements.

5 Related Work

Formal Verification of DNNs. The formal verification of DNNs has received growing attention [2, 3] due to their increasing use in safety-critical domains. Early efforts focused on fully-connected and

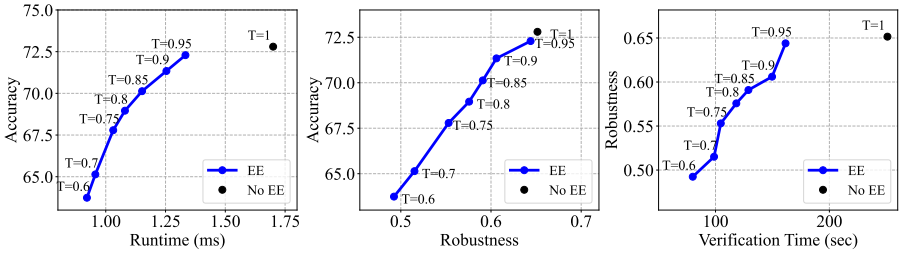


Figure 5: Impact of threshold selection on accuracy vs. runtime (left), accuracy vs. robustness (middle), and robustness vs. verification time (right) for CNN on CIFAR-10, $\epsilon = 0.005$.

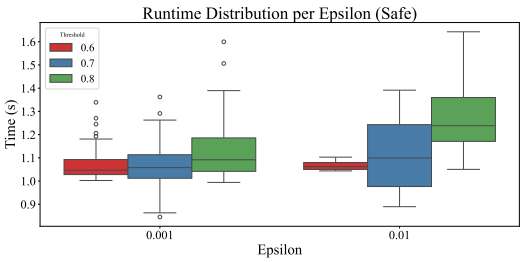


Figure 6: Impact of threshold on verification time for CNN on CIFAR-10.

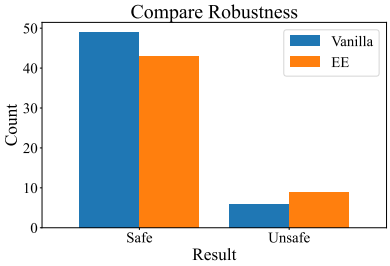


Figure 7: Robustness comparison between vanilla and EE-augmented models.

convolutional networks [4, 17, 20], while more recent work has expanded to RNNs [22], LSTMs [31], transformers [40], and GNNs [24, 50].

Various techniques have been proposed to improve scalability, including symbolic propagation, abstract interpretation, abstraction-refinement, adversarial pruning, certified training, and training for verification [6, 12, 14, 15, 28, 29, 32, 34, 42, 46–48, 53–55]. *PAVE*, which adds lightweight EE heads post-hoc, is *orthogonal* to these techniques: improved verifiers or architectures directly improve *PAVE* too.

Early Exit Architectures. EE represents a dynamic inference strategy within the broader landscape of DNN optimization, which also includes quantization [5], pruning [16], knowledge distillation [36], and neural architecture search [57]. Models with EEs accelerate inference by allowing early termination for easy inputs [1, 8, 33, 38]. Our work is, to our knowledge, the first to explore how EEs can be exploited for scalable formal verification of neural networks.

6 Conclusion and Future Work

Conclusion. We introduced *PAVE*, a paradigm that exploits EE architectures to solve the large majority of robustness queries in a fraction of the time by decomposing hard DNN verification problems into tractable sub-problems. We showed that EEs can be added to many standard DNNs without accuracy degradation, formalized the EE robustness property and proved its FPT complexity, and proposed a sound and complete algorithm with two optimizations that achieve up to $10\times$ speedup on `SAFE` cases. Experiments demonstrate that *PAVE* enables verification of queries that are otherwise intractable, and provides actionable design guidance on EE hyperparameters.

Future Work. Several directions remain open, including evaluation on larger and non-image models, extension to other properties, distributed verification across exits, and alternative exit conditions [33].

7 Acknowledgements

The work of Elboher and Katz was partially funded by the European Union (RobustifAI project, ID 101212818). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HADEA). Neither the European Union nor the granting authority can be held responsible for them. The work of Raviv and Kugler was supported by the Horizon 2020 research and innovation programme for the Bio4Comp project under grant agreement number 732482, by the Israel Science Foundation (Grant No. 190/19) and by the Data Science Institute at Bar-Ilan University (seed grant).

References

[1] D. Bajpai and M. Hanawal. A Survey of Early Exit Deep Neural Networks in NLP, 2025. Technical Report. https://arxiv.org/abs/2501.07670.

[2] C. Brix, M. Müller, S. Bak, T. Johnson, and C. Liu. First Three Years of the International Verification of Neural Networks Competition (VNN-COMP). *Journal on Software Tools for Technology Transfer (STTT)*, pages 1–11, 2023.

[3] C. Brix, S. Bak, T. Johnson, and H. Wu. The Fifth International Verification of Neural Networks Competition (VNN-COMP 2024): Summary and Results, 2024. Technical Report. https://arxiv.org/abs/2412.19985.

[4] R. Bunel, I. Turkaslan, P. Torr, M. Kumar, J. Lu, and P. Kohli. Branch and Bound for Piecewise Linear Neural Network Verification. *Journal on Machine Learning Research (JMLR)*, 2020.

[5] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A Survey of Model Compression and Acceleration for Deep Neural Networks, 2017. Technical Report. https://arxiv.org/abs/1710.09282.

[6] E. Cohen, Y. Y. Elboher, C. Barrett, and G. Katz. Tighter Abstract Queries in Neural Network Verification. In *Proc. 24th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pages 124–143, 2023.

[7] D. Dalrymple, J. Skalse, Y. Bengio, S. Russell, M. Tegmark, S. Seshia, S. Omohundro, C. Szegedy, B. Goldhaber, N. Ammann, et al. Towards Guaranteed Safe AI: A Framework for Ensuring Robust and Reliable AI Systems, 2024. Technical Report. https://arxiv.org/abs/2405.06624.

[8] A. Dimitriou, L. Xun, J. Hare, and G. Merrett. Realisation of Early-Exit Dynamic Neural Networks on Reconfigurable Hardware. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2024.

[9] Y. Dong. Resource-Constrained Edge AI with Early Exit Prediction. *Journal of Communications and Information Networks (JCIN)*, pages 122–134, 2022.

[10] R. Downey. A Parameterized Complexity Tutorial. In *Language and Automata Theory and Applications*, pages 38–56, 2012.

[11] H. Duong, T. Nguyen, and M. Dwyer. A dpll(t) framework for verifying deep neural networks, 2024. Technical Report. https://arxiv.org/abs/2307.10266.

[12] H. Duong, D. Xu, T. Nguyen, and M. Dwyer. Harnessing Neuron Stability to Improve DNN Verification. *Proc. ACM on Software Engineering (PACMSE)*, pages 859–881, 2024.

[13] R. Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Proc. Conf. on Automated Technology for Verification and Analysis (ATVA)*, 2017.

[14] Y. Elboher, J. Gottschlich, and G. Katz. An Abstraction-Based Framework for Neural Network Verification. In *Proc. 32nd Int. Conf. Computer Aided Verification (CAV)*, pages 43–65, 2020.

[15] Y. Elboher, E. Cohen, and G. Katz. Neural Network Verification Using Residual Reasoning. In *Proc. 20th Conf. on Software Engineering and Formal Methods (SEFM)*, pages 173–189, 2022.

[16] J. Frankle and M. Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, 2019. Technical Report. https://arxiv.org/abs/1803.03635.

[17] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. *Proc. 39th IEEE Symposium on Security and Privacy (S&P)*, pages 3–18, 2018.

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proc. 34th IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[19] S. Huang, A. Pareek, M. Jensen, M. Lungren, S. Yeung, and A. Chaudhari. Self-Supervised Learning for Medical Image Classification: a Systematic Review and Implementation Guidelines. *NPJ Digital Medicine*, page 74, 2023.

[20] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. Computer Aided Verification (CAV)*, pages 97–117, 2017.

[21] L. Khachiyan. Polynomial Algorithms in Linear Programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[22] I. Khmelnitsky, D. Neider, R. Roy, B. Barbot, B. Bollig, A. Finkel, S. Haddad, M. Leucker, and L. Ye. Property-Directed Verification of Recurrent Neural Networks, 2020. Technical Report. https://arxiv.org/abs/2009.10610.

[23] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images, 2009. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[24] T. Ladner, M. Eichelbeck, and M. Althoff. Formal Verification of Graph Convolutional Networks with Uncertain Node Features and Uncertain Graph Structure. *Transactions on Machine Learning Research (TMLR)*, 2025.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE*, pages 2278–2324, 1998.

[26] Y. LeCun, C. Cortes, and C. Burges. MNIST Handwritten Digit Database. *ATT Labs*, 2010.

[27] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, M. Kochenderfer, et al. Algorithms for Verifying Deep Neural Networks. *Foundations and Trends in Optimization (FnT Opt)*, pages 244–404, 2021.

[28] Z. Liu, Z. Zhao, F. Song, J. Sun, P. Yang, X. Huang, and L. Zhang. Training Verification-Friendly Neural Networks via Neuron Behavior Consistency. In *Proc. 39th Conf. Assoc. for Advancements of Artificial Intelligence (AAAI)*, pages 5757–5765, 2025.

[29] Y. Mao, M. Mueller, M. Fischer, and M. Vechev. Connecting Certified and Adversarial Training. In *Proc. 37th Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2023.

[30] Y. Mao, S. Balauca, and M. Vechev. CTBENCH: A Library and Benchmark for Certified Training, 2025. Technical Report. https://arxiv.org/abs/2406.04848.

[31] F. Moradkhani, C. Fibich, and M. Fränzle. Verification of LSTM Neural Networks with Non-linear Activation Functions. In *Proc. Conf. on NASA Formal Methods (NFM)*, pages 1–15, 2023.

[32] M. Mueller, F. Eckert, M. Fischer, and M. Vechev. Certified Training: Small Boxes are All You Need. In *Proc. 11th Int. Conf. on Learning Representations (ICLR)*, 2023.

[33] H. Rahmath P, V. Srivastava, K. Chaurasia, R. G. Pacheco, and R. S. Couto. Early-Exit Deep Neural Network — A Comprehensive Survey. *ACM Computing Surveys*, 2024.

[34] A. Palma, R. Bunel, K. Dvijotham, M. Kumar, R. Stanforth, and A. Lomuscio. Expressive Losses for Verified Robustness via Convex Combinations. In *Proc. 12th Int. Conf. on Learning Representations (ICLR)*, 2024.

[35] P. Panda, A. Sengupta, and K. Roy. Conditional Deep Learning for Energy-Efficient and Enhanced Pattern Recognition. In *Proc. 19th Conf. on Design, Automation and Test in Europe (DATE)*, pages 475–480, 2016.

[36] M. Phuong and C. Lampert. Distillation-Based Training for Multi-Exit Architectures. In *Proc. 33rd IEEE/CVF Int. Conf. on Computer Vision (ICCV)*, pages 1355–1364, 2019.

[37] S. Russell. Provably Beneficial Artificial Intelligence. In *Proc. 27th Int. Conf. on Intelligent User Interfaces (IUI)*, page 3, 2022.

[38] E. Samikwa, A. Maio, and T. Braun. Adaptive Early Exit of Computation for Energy-Efficient and Low-Latency Machine Learning over IoT Networks. In *Proc. 19th IEEE Annual Consumer Communications and Networking Conf. (CCNC)*, pages 574–580, 2022.

[39] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini. Why Should We Add Early Exits to Neural Networks? *Cognitive Computation*, pages 954–966, 2020.

[40] Z. Shi, H. Zhang, K. Chang, M. Huang, and C. Hsieh. Robustness Verification for Transformers, 2020. Technical Report. https://arxiv.org/abs/2002.06622.

[41] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proc. 3rd Int. Conf. on Learning Representations (ICLR)*, 2015.

[42] G. Singh, T. Gehr, M. Püschel, and M. Vechev. An Abstract Domain for Certifying Neural Networks. *Proc. 3rd ACM on Programming Languages (PACMPL)*, pages 1–30, 2019.

[43] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing Properties of Neural Networks, 2014. Technical Report. https://arxiv.org/abs/1312.6199.

[44] S. Teerapittayanon, B. McDanel, and H. Kung. Branchynet: Fast Inference Via Early Exiting from Deep Neural Networks. In *Proc. 23rd Int. Conf. on Pattern Recognition (ICPR)*, pages 2464–2469, 2016.

[45] V. Tjeng, K. Xiao, and R. Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *Proc. 7th Int. Conf. on Learning Representations (ICLR)*, 2019.

[46] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient Formal Safety Analysis of Neural Networks. *Proc. 31st Int. Conf. on Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[47] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal Security Analysis of Neural Networks Using Symbolic Intervals. In *Proc. 27th USENIX Conference on Security Symposium*, pages 1599–1614, 2018.

[48] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C. Hsieh, and J. Kolter. Beta-crown: Efficient Bound Propagation with Per-Neuron Split Constraints for Neural Network Robustness Verification. *Proc. 34th Conf. on Advances in Neural Information Processing Systems (NeurIPS)*, pages 29909–29921, 2021.

[49] X. Wang, Y. Luo, D. Crankshaw, A. Tumanov, F. Yu, and J. Gonzalez. Idk cascades: Fast Deep Learning by Learning Not to Overthink, 2017. Technical Report. https://arxiv.org/abs/1706.00885.

[50] H. Wu, C. Barrett, M. Sharif, N. Narodytska, and G. Singh. Scalable Verification of GNN-Based job schedulers. *Proc. 6th ACM Conf. on Programming Languages (PACMPL)*, 2022.

[51] J. Xin, R. Tang, Y. Yu, and J. Lin. BERxiT: Early Exiting for BERT with Better Fine-Tuning and Extension to Regression. In *Proc. 16th Conf. European Chapter of the Association for Computational Linguistics (EACL)*, pages 91–104, 2021.

[52] K. Xu, Z. Shi, H. Zhang, Y. Wang, K. Chang, M. Huang, B. Kailkhura, X. Lin, and C. Hsieh. Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. *Proc. 34th Conf. on Advances in Neural Information Processing Systems (NeurIPS)*, pages 1129–1141, 2020.

[53] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C. Hsieh. Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers. In *Proc. 9th Int. Conf. on Learning Representations (ICLR)*, 2021.

[54] H. Zhang, S. Wang, K. Xu, L. Li, B. Li, S. Jana, C. Hsieh, and J. Kolter. General Cutting Planes for Bound-Propagation-Based Neural Network Verification. In *Proc. 35th Conf. on Advances in Neural Information Processing Systems (NeurIPS)*, pages 1656–1670, 2022.

[55] D. Zhou, C. Brix, G. Hanasusanto, and H. Zhang. Scalable Neural Network Verification with Branch-and-bound Inferred Cutting Planes. In *Proc. 37th Conf. on Advances in Neural Information Processing Systems (NeurIPS)*, pages 29324–29353, 2024.

[56] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei. BERT Loses Patience: Fast and Robust Inference with Early Exit, 2020. Technical Report. https://arxiv.org/abs/2006.04152.

[57] B. Zoph and Q. Le. Neural Architecture Search with Reinforcement Learning, 2017. Technical Report. https://arxiv.org/abs/1611.01578.

A An Example of a Neural Network with Early Exits

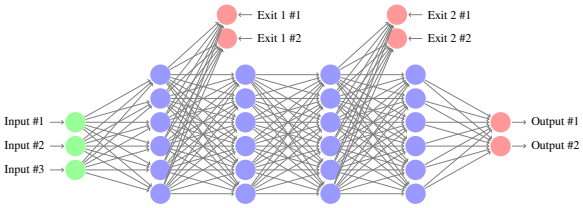


Figure 8: A fully connected DNN with two EEs placed at the first and third layers. The neuron values at these exits are the results of applying SoftMax to the hidden values in the corresponding layers.

B Complexity Analysis

B.1 Alg. 1's Complexity

We analyze the complexity of the proposed verification algorithms. Solving verification queries is NP-Hard [20], and current methods have exponential complexity in the network's number of neurons in a worst-case scenario. We denote the worst-case complexity of the underlying verification tool by $\mathcal{O}(2^N)$ where N is the number of neurons in $\mathcal{N}$.

Theorem 5. *The worst-case complexity of Alg. 1 is* $\mathcal{O}(2^N)$.

Proof. Alg. 1 applies $E \cdot (C-1)+1$ verification queries, where E is the number of EEs and C is the number of classes ($+1$ for the last query). Each query is $\mathcal{O}(2^{N_i})$ where N_i is the number of neurons in the partial network until the i'th exit (including), summing to an overall $\mathcal{O}(2^N)$ complexity. □

B.2 Alg. 2's Complexity

Alg. 2 performs break and continue heuristic optimizations, which are not necessarily triggered, and hence its worst-case complexity is similar to that of Alg. 1. However, its complexity decreases under the trace stability assumption. We prove Thm. 4:

Theorem. *Given a network* $\mathcal{N}$ *with EEs and ReLU activations, layer width bound* k*, input* x*, and* $\epsilon > 0$*, if* $\tau(x)$ *is stable in* B_ϵ^x*, then Alg. 2 runtime is* $\mathcal{O}(2^{k\cdot|\tau(x)|}) \cdot poly$*(#neurons in* $\mathcal{N}$*).*

Proof. We denote the index of the early exit where the output of x is returned by E_x. In Alg. 2, line 8 checks if the winner always wins in the current exit. If the trace of each input in B_ϵ^x equals $\tau(x)$ and the result is `SAFE`, it must be returned in line 8 in the E_x'th iteration: it cannot be returned before since x itself constitutes a counterexample, and it is returned at iteration E_x since the trace of all inputs equals $\tau(x)$. If the answer is `UNSAFE`, the counterexample must be found by the E_x'th iteration; in each iteration, line 14 launches $ExistsPrevCEX(\mathcal{N}, x, \epsilon, k)$, which checks all possible counterexamples until the E_x'th exit. Because $\tau(x') = \tau(x)$ for all inputs, all possible behaviors are examined after the E_x'th iteration.

This means that Alg. 2 will not proceed to iterations that run verification queries on sub-networks larger than the sub-network that x was propagated through. The number of neurons in each such sub-network is no more than $k \cdot |\tau(x)|$, and solving verification queries of that size takes $2^{k\cdot|\tau(x)|} \cdot poly$(#neurons in $\mathcal{N}$). Therefore, the complexity is bounded by $\mathcal{O}(2^{k\cdot|\tau(x)|}) \cdot poly$(#neurons in $\mathcal{N}$). □

C Proofs

In this appendix we provide the detailed proofs for the algorithms.

Algorithm 3 Pareto Verification – Break Optimization

Input $\mathcal{N}$, x, ϵ_p **Output** $\mathcal{N}$ is robust in $B_\epsilon(x)$, or counterexample

1: $w = argmax(\mathcal{N}(x))$
2: ee := indices of layers with early exits in $\mathcal{N}$
3: last := index of last layer in $\mathcal{N}$
4: **for** $k \in ee \cup \{last\}$ **do**
5: **if** k $\neq$ last **then**
6: $\mathcal{P} := \exists x' \in B_\epsilon^x : \mathcal{N}(x')_w^k < T$
7: **else**
8: $\mathcal{P} := \exists x' \in B_\epsilon^x, \exists i \in \mathcal{C} \setminus \{w\} : \mathcal{N}(x')_w < \mathcal{N}(x')_i$
9: **end if**
10: res, cex = Verify($\mathcal{N}, B_\epsilon^x, \mathcal{P}$)
11: **if** res == SAFE **then**
12: **return** SAFE
13: **end if**
14: res, cex = $ExistsPrevCEX(\mathcal{N}, x, \epsilon, k)$
15: **if** res == UNSAFE **then**
16: **return** UNSAFE, cex
17: **end if**
18: **end for**
19: **return** SAFE

C.1 Proof of Thm. 1

Theorem. *If the underlying verification tool is sound and complete, Alg. 1 is sound and complete.*

Proof. We split the proof for soundness into 3 parts:

1. There is a satisfying example to P_{ee} if and only if there are exit $k \in ee \cup \{last\}$ and runner-up $i \in \mathcal{C} \setminus \{w\}$ such that the runner-up wins at exit k and there is no preceding layer where the winner wins. This is encoded in line 5 (for early exits) and line 7 (for the last exit) in Alg. 1.

2. If Alg. 1 returns UNSAFE, one of the properties in lines 5 or 7 is UNSAFE. The result is sound if the underlying verification tool is sound.

3. Otherwise, if Alg. 1 does not return UNSAFE, it returns SAFE at the last line. Since the algorithm iterates over all exits and all runner-ups, completeness of the underlying tool implies that if there is no counterexample then P_{ee} is not satisfiable.

Completeness follows from the fact that there are finitely many queries, each guaranteed to terminate by completeness of the underlying tool. □

C.2 Proof of Thm. 2

Theorem. *Given a network $\mathcal{N}$ with EEs and ReLU activations, layer width bound k, input x, and $\epsilon > 0$, if $\tau(x)$ is stable in B_ϵ^x, then solving P_{ee} with $(\mathcal{N}, x, \epsilon)$ is FPT$(k \cdot |\tau(x)|)$.*

Proof. Given that $\tau(x)$ is stable in B_ϵ^x, every input in B_ϵ^x exits at the same exit as $\mathcal{N}(x)$. The verification process therefore only needs to consider the layers up to that exit. The number of neurons up to this exit is bounded by $k \cdot |\tau(x)|$. Solving the query is done by splitting each ReLU into two cases—active $(y = x)$ and inactive $(y = 0)$—giving $2^{k \cdot |\tau(x)|}$ combinations, each solvable in $poly$(#neurons in $\mathcal{N}$) via linear programming [21]. Therefore, the problem is in *FPT*$(k \cdot |\tau(x)|)$. □

C.3 Proof of Thm. 3

We separate the break and continue optimizations into Alg. 3 (blue lines only) and Alg. 4 (orange lines only), prove soundness and completeness for each by equivalence to Alg. 1, and then derive the correctness of Thm. 3.

Algorithm 4 Pareto Verification – Continue Optimization

Input $\mathcal{N}$, x, ϵ_p **Output** $\mathcal{N}$ is robust in $B_\epsilon(x)$, or counterexample

1: $w = argmax(\mathcal{N}(x))$
2: ee := indices of layers with early exits in $\mathcal{N}$
3: last := index of last layer in $\mathcal{N}$
4: **for** $k \in ee \cup \{last\}$ **do**
5: res, cex = Verify($\mathcal{N}, B_\epsilon^x, \exists x' \in B_\epsilon^x : \mathcal{N}_w^k(x') < 1 - T$)
6: **if** k == last $\vee$ res == `UNSAFE` **then**
7: res, cex = $ExistsPrevCEX(\mathcal{N}, x, \epsilon, k)$
8: **if** res == `UNSAFE` **then**
9: **return** `UNSAFE`, cex
10: **end if**
11: **end if**
12: **end for**
13: **return** `SAFE`

Theorem 6. *Alg. 4 is equivalent to Alg. 1.*

Proof. The additional logic in Alg. 4 (line 5) checks whether the winner's value might be smaller than $1 - T$. If not, no runner-up can exceed T (scores sum to 1), so the result of $ExistsPrevCEX$ must be `SAFE` and the inner loop can be skipped. If the condition is satisfied, or at the last layer (where the condition gives no assurance about runner-ups), the verification proceeds as in Alg. 1. In both cases the result is equivalent. □

Theorem 7. *Alg. 3 is equivalent to Alg. 1.*

Proof. Alg. 3 adds a check at each iteration (line 10) for whether the winner might not win in the current exit. If `SAFE` is returned (line 11 holds), verification halts and returns `SAFE`. This is valid because for all subsequent exits, neither line 5 nor line 7 of Alg. 1 would hold—since there is a preceding exit where the winner wins for every example. If the condition is satisfiable, Alg. 3 continues to check for runner-up counterexamples, exactly as Alg. 1 does, yielding equivalent results. □

Theorem. *Alg. 2 is equivalent to Alg. 1.*

Proof. Alg. 2 sequentially incorporates both Alg. 4 and Alg. 3. Correctness follows from Thm. 6 and Thm. 7. □

D Datasets and Models Technical Details

Below we describe each dataset and architecture, summarize key characteristics in Tab. 4, and present the training protocol.

We used three datasets:

- **MNIST** [26]: Grayscale handwritten digits, 10 classes.
- **CIFAR-10** [23]: Color images, 10 classes; a standard benchmark for formal verification.
- **CIFAR-100** [23]: Extension of CIFAR-10 with 100 classes.

Table 4: Metadata for datasets used in the evaluation.

Dataset	Train Size	Test Size	#Classes	Input Shape
MNIST	60,000	10,000	10	$1 \times 28 \times 28$
CIFAR-10	50,000	10,000	10	$3 \times 32 \times 32$
CIFAR-100	50,000	10,000	100	$3 \times 32 \times 32$

For each dataset, we trained models with EEs:

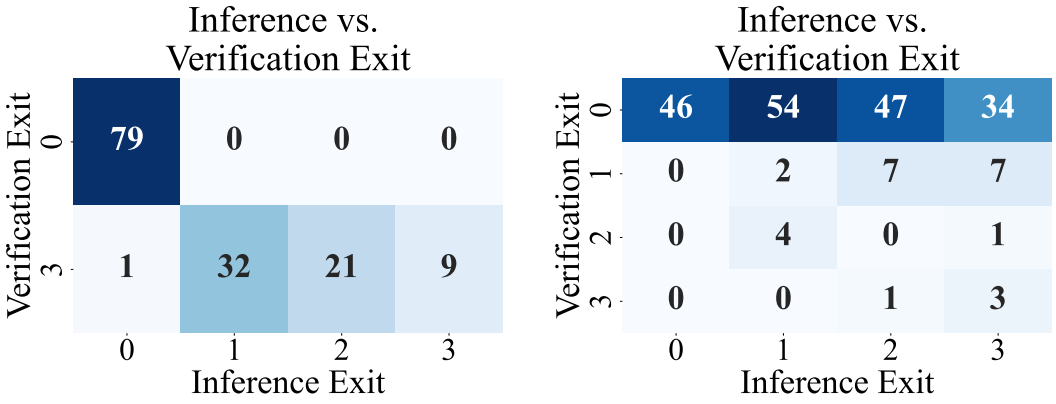
- **FC-6**: A fully connected architecture with 6 layers; first three equipped with EEs. Trained on MNIST.
- **LeNet-5 (CNN)** [25]: 2 convolutional + 3 linear layers; EEs after first and second convolutional layers. Trained on CIFAR-10.
- **Modified ResNet-18** [18]: ResNet-18 with MaxPool replaced by AveragePool; EEs after the first and second blocks. Trained on CIFAR-10.
- **VGG-16** [41]: Standard VGG-16 (13 convolutional layers, with Adaptive Average Pool instead of MaxPool for verifier compatibility); EEs after the 6th and 10th convolutional layers. Trained on CIFAR-10.

Training Protocol. All models are trained using standard supervised learning. For CIFAR-10/100, we apply data augmentation (random cropping, horizontal flipping) and normalize with dataset-specific statistics. **MNIST** models are trained for 10 epochs (SGD, learning rate reduced $\times 0.1$ every 4 epochs). **CIFAR-10 models** (ResNet-18, VGG-16): 200 epochs, SGD with momentum 0.9 and weight decay 5×10^{-4}, step LR schedule at epochs 100 and 150. **LeNet-5**: 30 epochs, Adam, initial LR 0.001, decayed at epochs 10 and 20. **CIFAR-100 models**: Adam, LR 0.001, cosine annealing over 200 epochs. For EE heads: freeze backbone, train each exit sequentially for 20 epochs (10 for FC and LeNet) with the same optimizer and two $\times 0.1$ LR decays.

Technical Non-Triviality. While conceptually simple, our method has two non-trivial aspects. First, it introduces many additional verification queries in the worst case, and its effectiveness depends on whether early exits succeed in verification; performance gains are not guaranteed and are validated only empirically. Second, each partial verification query verifies a more complex property involving the exit condition with Softmax, making the problem strictly harder. The benefit arises only when combined with a state-of-the-art verifier supporting Softmax [48, 53].

Relation to Certified Training. Unlike certified training methods, which modify the training process to improve robustness, our method preserves standard training and augments the model with EEs to accelerate verification. The two approaches are orthogonal and can be combined. For comparison: certified CNN-7 with $\epsilon \approx 0.0078$ achieves accuracy 78.82% and robustness 64.41% [30]; our EE CNN-5 with $\epsilon = 0.005$ and $T = 0.9$ achieves accuracy 71.34% and robustness 60.61% with a smaller network, demonstrating that our fine-tuning preserves robustness while providing efficiency gains.

E Verification Exit Versus Inference Exit: MNIST & FC-6



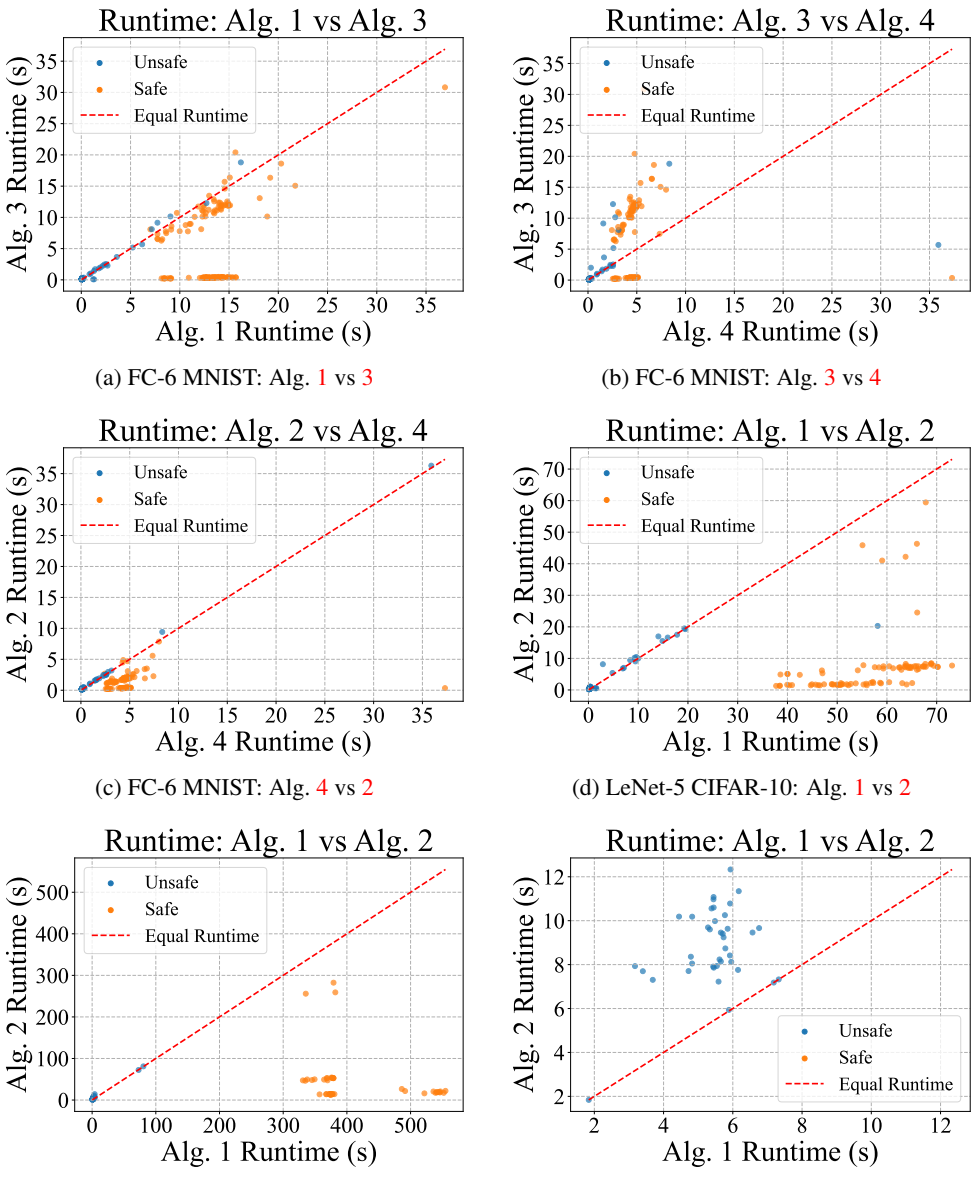
(e) FC-6 `SAFE` (f) FC-6 `UNSAFE`

Figure 9: Heatmaps of inference vs. verification exit layer for FC-6 and MNIST. As with the CIFAR-10 models, a strong diagonal correlation is observed for `SAFE` cases.

F Ablation Study: Algorithm Variants

We compare Alg. 1 and Alg. 2 with each optimization applied individually. Fig. 10 compares algorithm pairs; axes represent runtime (seconds); points are color-coded by result.

Subfigures (a)–(c) ablate on MNIST: (a) Break-only (Alg. 3) outperforms the baseline; (b) Continue-only (Alg. 4) outperforms Break-only; (c) the combined algorithm (Alg. 2) is the best. Subfigures (d)–(f) compare the baseline and combined algorithm on LeNet-5, ResNet-18, and VGG-16, showing consistent superiority in `SAFE` cases and equal runtime on `UNSAFE` cases.



(a) FC-6 MNIST: Alg. 1 vs 3

(b) FC-6 MNIST: Alg. 3 vs 4

(c) FC-6 MNIST: Alg. 4 vs 2

(d) LeNet-5 CIFAR-10: Alg. 1 vs 2

(e) ResNet-18 CIFAR-10: Alg. 1 vs 2

(f) VGG-16 CIFAR-10: Alg. 1 vs 2

Figure 10: Ablation study comparing algorithm variants across benchmarks.

G Extendability to Other Properties and Activations

While we instantiated *PAVE* for local robustness, the formulation extends naturally to other property types and activation functions.

Other Verification Properties. At each exit k, the robustness clause can be replaced by an arbitrary predicate ϕ over network outputs:

$$\exists x' \in B_\varepsilon(x) : \phi(N_k(x')) \wedge \forall e < k : N_e^w(x') < T_e,$$

where the right-hand term enforces the EE semantics. Examples include safety constraints ($\phi \equiv g(N_k(x')) \leq 0$) and fairness predicates. The Break optimization generalizes by substituting the class-margin test with the corresponding sufficient condition for the new property.

Activation Functions. Our framework is activation-independent. The FPT result (Thm. 2) assumes piecewise-linear activations, but soundness and completeness of the general formulation hold as long as the underlying verifier supports the activation functions used.

H Encoding Details and Choice of Underlying Verifier

In this appendix we elaborate on the encoding of our verification queries and explain our choice of underlying verifier.

Structure of our queries. Each verification query produced by Alg. 1 or Alg. 2 is a conjunction of constraints over the network outputs at *multiple* exit layers. Concretely, at exit k, we must simultaneously assert that (i) a runner-up wins at exit k, and (ii) for every preceding exit $e < k$ with an EE, the winner's SoftMax score is below the threshold T_e. Expressing this natively requires a verifier that supports nested conjunctions ($\wedge$) over multiple output heads together with the SoftMax used by the EE confidence function.

On the choice of α-β-CROWN as the sole backend. α-β-CROWN is a state-of-the-art verifier that handles SoftMax robustly enough to support our queries at scale. SoftMax support is essential for our setting: without it, the threshold condition on an early exit cannot be encoded at all, since it is defined on post-SoftMax probabilities. We evaluated two alternatives: Marabou can encode SoftMax in principle, but stalls under full branch-and-bound even on the first early-exit query — a limitation confirmed by the Marabou maintainers. NeuralSAT lacks SoftMax support entirely [11], making it incompatible with our encoding without tool-level modifications. As SoftMax support matures in other verifiers, our framework requires no algorithmic changes to adopt them.

Our encoding. To circumvent the lack of nested AND support in α-β CROWN, we decompose each multi-conjunct query into a collection of smaller single-conjunct queries and combine their results post hoc in a sound manner (a conjunction is `UNSAFE` only if each conjunct is individually satisfiable under the same witness, which we enforce by the structure of the loop in Alg. 1). To avoid biasing the comparison, we apply the *same* decomposition when running the baseline (no-EE) verification queries. This guarantees a fair head-to-head comparison and isolates the effect of early exits on verification time. We consulted the α-β-CROWN maintainers, who confirmed that it is an implementation limitation – theoretically it seems to be possible to support but currently the code only supports relatively simple specifications.

Future simplifications. Our decomposition is a workaround, not a fundamental requirement. Once a verifier supporting both SoftMax and native nested ANDs becomes available (e.g., via an extension of Marabou or α-β CROWN), each query in Alg. 1 could be issued as a single call, removing encoding overhead and further reducing runtime. Our framework is verifier-agnostic and would directly benefit from such improvements.

I Why Trace Stability Does Not Hold for Unsafe Queries

Recall that the trace $\tau(x)$ of an input x is the set of layers x traverses before an EE condition fires, and $\tau(x)$ is *stable* in B_ϵ^x if $\tau(x') = \tau(x)$ for every $x' \in B_\epsilon^x$ (see Sec. 3.3). In Fig. 4, we observed that trace stability holds in practice for the large majority of inputs that yield `SAFE` results, but breaks down on inputs that yield `UNSAFE` results. Here we explain why this asymmetry is expected—and, importantly, why it does not compromise the soundness, completeness, or practical efficiency of *PAVE*.

SAFE inputs: stability is the typical regime. An input x for which local robustness holds in B_ϵ^x is, by definition, a "well-separated" input: the winner class dominates over the entire ball, so the SoftMax distribution changes only mildly under perturbation. In particular, the winner's SoftMax score tends to remain above the exit threshold T at the same layer at which x itself exits. Thus every $x' \in B_\epsilon^x$ produces the same trace as x, and trace stability holds. This is exactly what we observe empirically in Fig. 4(a,c): for `SAFE` cases, the inference exit and verification exit layers coincide on the diagonal.

UNSAFE inputs: stability cannot be expected to hold. An `UNSAFE` query is, by definition, witnessed by a perturbation $x' \in B_\epsilon^x$ whose predicted class differs from that of x. For the prediction to flip, the SoftMax distribution at *some* exit must change substantially: either (i) the original winner's score at an early exit drops below the threshold T, causing x' to not exit where x did, or (ii) a runner-up's score rises above T at an earlier exit, causing x' to exit earlier. In either case, $\tau(x') \neq \tau(x)$, so $\tau(x)$ is *not* stable in B_ϵ^x. Trace stability and the existence of an adversarial counterexample are thus fundamentally at odds: any counterexample is a direct witness of trace instability.

Why this is not a problem. The FPT guarantee of Thm. 2 is conditioned on trace stability and applies to `SAFE` instances, which is exactly where a tight complexity bound matters (`UNSAFE` instances are resolved as soon as a counterexample is found, typically in a single early-exit query—see the UNSAFE runtimes in Tab. 2). Soundness and completeness of Alg. 1 and Alg. 2 (Thms. 1 and 3) do *not* depend on trace stability at all, and thus hold on `UNSAFE` inputs regardless.

Empirical corroboration. The heatmaps in Fig. 4(b,d) and Fig. 9 show that `UNSAFE` queries are concentrated in the first verification exit: counterexamples, when they exist, are typically found early regardless of where the original input x would have exited. This matches the theoretical picture above: the counterexample takes a different path through the network than x, and our algorithm discovers it as soon as that path is explored.

J Disclosure: Use of Large Language Models (LLMs)

The authors were solely responsible for developing the research questions, designing the methodology, performing the analysis, and interpreting the findings. A large language model (LLM) was employed only to assist with improving the clarity and style of the writing, without influencing any substantive aspects of the research.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction clearly state all contributions: the 80/20 verification paradigm, the formal EE robustness property P_{ee}, the algorithm and two optimizations with soundness/completeness proofs, and experimental results on MNIST, CIFAR-10, and CIFAR-100.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: The Future Work paragraph in Sec. 6 identifies limitations (currently restricted to local robustness, not yet parallelized across exits, limited to specific confidence functions). Sec. 4.1 details the setup limitations regarding the use of the underlying verification tool. App. D additionally discusses the technical non-triviality of the approach, including that performance gains depend on early exits succeeding in practice. A separat "Limitations" section does not appear in the paper due to lack of space.

   Guidelines:

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

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

   Justification: All four theorems (soundness/completeness of both algorithms and two FPT complexity results) are stated with explicit assumptions and proved in full in App. C.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: Full training details (architectures, optimizers, learning rate schedules, data augmentation, EE training protocol) are provided in App. D. The underlying verifier (Alpha-Beta-CROWN) is identified with citations, and the hardware setup (NVIDIA A100 GPU) is reported in Sec. 4.1.

   Guidelines:

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

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

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

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

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

   Answer: [No]

   Justification: Code and trained models are not released at submission time to preserve anonymity. All datasets used (MNIST, CIFAR-10, CIFAR-100) are publicly available standard benchmarks. Code and models will be released upon acceptance.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Sec. 4.1 describes the experimental setup and hardware; App. D provides full training details including optimizers, learning rate schedules, data augmentation, and EE training protocol for all benchmarks.

   Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [Yes]

Justification: Table 2 reports mean, standard deviation, and median verification runtimes for each benchmark and algorithm, capturing variability across queries. The nature of the experiment (deterministic formal verification) means there is no training randomness to report separately.

Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Sec. 4.1 reports the machine used (NVIDIA A100 GPU, 80 GB HBM2e). Per-query runtime statistics (mean, std, median) are reported in Table 2 for all benchmarks. Individual query timeouts (1-hour and 2-hour) are noted in the text.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: This work develops formal verification methods to improve the safety of DNN deployments. It involves no human subjects, no sensitive data, and raises no ethical concerns.

   Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: This is foundational formal methods research. The work exclusively improves safety guarantees for DNN deployments in safety-critical domains; there is no plausible negative societal impact pathway for a method whose sole effect is making safety verification faster and more scalable.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper does not release any models or datasets with high risk for misuse. The models trained are standard image classifiers on public benchmarks (MNIST, CIFAR-10, CIFAR-100).

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: All datasets (MNIST, CIFAR-10, CIFAR-100) and architectures (FC-6, LeNet-5, ResNet-18, VGG-16) are properly cited. The underlying verifier Alpha-Beta-CROWN is cited with its associated papers. All assets are standard academic benchmarks distributed for research use.

    Guidelines:

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

13. **New assets**

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

    Answer: [No]

    Justification: The paper trains new EE-augmented model variants used in evaluation, but these are not released at submission time to preserve anonymity. Full documentation (architectures, training protocol) is provided in App. D; models will be released with code upon acceptance.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects; all experiments use standard computer vision benchmarks.

Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects; no IRB approval is required.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: LLMs are not used as part of the core methodology. The method relies on constraint-satisfaction-based formal verifiers, not language models. App. J discusses the use of LLMs in the creation of the manuscript.

    Guidelines:

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