


Efficient Neural Network Verification via Order Leading Exploration of Branch-and-Bound Trees

Guanqin Zhang ✉ 


University of New South Wales, Sydney, Australia
CSIRO's Data61, Sydney, Australia

Kota Fukuda ✉ 

Kyushu University, Fukuoka, Japan

Zhenya Zhang ✉ 


Kyushu University, Fukuoka, Japan
National Institute of Informatics, Tokyo, Japan

H.M.N. Dilum Bandara ✉ 

CSIRO's Data61, Sydney, Australia
University of New South Wales, Sydney, Australia

Shiping Chen ✉ 

CSIRO's Data61, Sydney, Australia
University of New South Wales, Sydney, Australia

Jianjun Zhao ✉ 

Kyushu University, Fukuoka, Japan

Yulei Sui ✉ 

University of New South Wales, Sydney, Australia

Abstract

The vulnerability of neural networks to adversarial perturbations has necessitated formal verification techniques that can rigorously certify the quality of neural networks. As the state-of-the-art, branch-and-bound (BaB) is a “divide-and-conquer” strategy that applies off-the-shelf verifiers to sub-problems for which they perform better. While BaB can identify the sub-problems that are necessary to be split, it explores the space of these sub-problems in a naive “first-come-first-served” manner, thereby suffering from an issue of inefficiency to reach a verification conclusion. To bridge this gap, we introduce an order over different sub-problems produced by BaB, concerning with their different likelihoods of containing counterexamples. Based on this order, we propose a novel verification framework Oliva that explores the sub-problem space by prioritizing those sub-problems that are more likely to find counterexamples, in order to efficiently reach the conclusion of the verification. Even if no counterexample can be found in any sub-problem, it only changes the order of visiting different sub-problems and so will not lead to a performance degradation. Specifically, Oliva has two variants, including Oliva^{GR}, a greedy strategy that always prioritizes the sub-problems that are more likely to find counterexamples, and Oliva^{SA}, a balanced strategy inspired by simulated annealing that gradually shifts from exploration to exploitation to locate the globally optimal sub-problems. We experimentally evaluate the performance of Oliva on 690 verification problems spanning over 5 models with datasets MNIST and CIFAR-10. Compared to the state-of-the-art approaches, we demonstrate the speedup of Oliva for up to 25× in MNIST, and up to 80× in CIFAR-10.

2012 ACM Subject Classification Software and its engineering → Formal software verification; Software and its engineering → Software testing and debugging

Keywords and phrases neural network verification, branch and bound, counterexample potentiality, simulated annealing, stochastic optimization

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2025.36

Supplementary Material *Software*: <https://github.com/DeepLearningVerification/Oliva>
Software (ECOOP 2025 Artifact Evaluation approved artifact):
<https://doi.org/10.4230/DARTS.11.2.11>

Funding Guanqin Zhang: CSIRO's Data61 PhD Scholarship

Kota Fukuda: JST BOOST Grant No. JPMJBS2406

Zhenya Zhang: JST BOOST Grant No. JPMJBY24D7 and JSPS Grant No. JP25K21179

Jianjun Zhao: JSPS KAKENHI Grant No. JP23H0337

Yulei Sui: Australian Research Council Grants No. FT220100391 and No. DP250101396



© Guanqin Zhang, Kota Fukuda, Zhenya Zhang, H.M.N. Dilum Bandara, Shiping Chen, Jianjun Zhao, and Yulei Sui;
licensed under Creative Commons License CC-BY 4.0

39th European Conference on Object-Oriented Programming (ECOOP 2025).

Editors: Jonathan Aldrich and Alexandra Silva; Article No. 36; pp. 36:1–36:29



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

The rapid advancement of artificial intelligence (AI) has propelled the state-of-the-art across various fields, including computer vision, natural language processing, recommendation systems, etc. Recently, there is also a trend that adopts AI products in safety-critical systems, such as autonomous driving systems, in which neural networks are used in the perception module to perceive external environments. In this type of application, unexpected behaviors of neural networks can bring catastrophic consequences and intolerable social losses; given that neural networks are notoriously vulnerable to deliberate attacks or environmental perturbations [28, 18], effective quality assurance techniques are necessary in order to expose their defects timely before their deployment in the real world.

Formal verification is a rigorous approach that can ensure the quality of target systems by providing mathematical proofs on conformance of the systems with their desired properties. In the context of neural networks, formal verification aims to certify that a neural network, under specific input conditions, will never violate a pre-defined specification regarding its behavior, such as safety and robustness, thereby providing a rigorous guarantee that the neural network can function as expected in real-world applications. With the growing emphasis on AI safety, neural network verification has emerged as a prominent area of research, leading to the development of innovative methodologies and tools.

As a straightforward approach, exact encoding solves the neural network verification problem by encoding the inference process and specifications to be logical constraints and applying off-the-shelf or dedicated solvers, such as MILP solvers [47] and SMT solvers [23], to solve the problem. However, this approach can be very slow due to the non-linearity of neural network inferences and so they are not scalable to neural networks of large sizes. In contrast, approximated methods [44, 41, 49] that over-approximate the reachable region of neural networks are more efficient: once the over-approximation satisfies the specification, the original output must also satisfy; however, if the over-approximation violates the specification, it does not indicate that the original output also violates, and in this case it may raise a *false alarm*, i.e., a specification violation that actually does not exist.

To date, *Branch-and-Bound* (BaB) [5] is the state-of-the-art verification approach that overarches multiple advanced verification tools, such as $\alpha\beta$ -Crown [51] and Marabou [52]. It is essentially a “divide-and-conquer” strategy, and is often jointly used with off-the-shelf approximated verifiers due to their great efficiency. Given a verification problem, it first applies an approximated verifier to the problem, and splits it to sub-problems if the verifier raises a false alarm, until all the sub-problems are verified or a real counterexample is detected, as a witness of specification violation. As application of approximated verifiers to sub-problems leads to less over-approximation, BaB can thus resolve the issue of false alarms of approximated verifiers and outperform their plain application to the original problem.

Motivations

As a “divide-and-conquer” strategy, BaB can produce a large space that consists of quantities of sub-problems, especially for verification tasks that are reasonably difficult. However, when exploring this space, the classic BaB adopts a naive “first come, first served” strategy, which ignores the importance of different sub-problems and thus is not efficient to reach a verification conclusion. Notably, different sub-problems produced by BaB are not equally important – with a part of the sub-problems, it is more likely to find a real counterexample that can show the violation of the specification, and thereby we can reach a conclusion for the verification efficiently without going through the remaining sub-problems.

Contributions

In this paper, we propose a novel verification approach *Oliva*, which is an order leading intelligent verification technique for artificial neural networks.

We first define an order called *counterexample potentiality* over the different sub-problems produced by BaB. Our order estimates *how likely* a sub-problem is to contain a counterexample, based on the following two attributes: **1.** the level of problem splitting of the sub-problem, which implies how much approximation the approximated verifier needs to perform. The less approximation there is, the more likely the verifier can find a real counterexample; **2.** a quantity obtained by applying approximated verifiers to the sub-problem, which is an indicator of *the degree* to which the neural network satisfies/violates the specification. By these two attributes, we define the counterexample potentiality order over the sub-problems, as a proxy to suggest their likelihood of containing counterexamples and serve as guidance for our verification approach.

Then, we devise our approach *Oliva* that exploits the order to explore the sub-problem space. In general, *Oliva* favors the sub-problems that are more likely to contain counterexamples. Once it can find a real counterexample, it can immediately terminate the verification and return with a verdict that the specification can be violated; even if it cannot find such a counterexample, after visiting all the sub-problems, it can still manage to certify the neural network without a significant performance degradation.

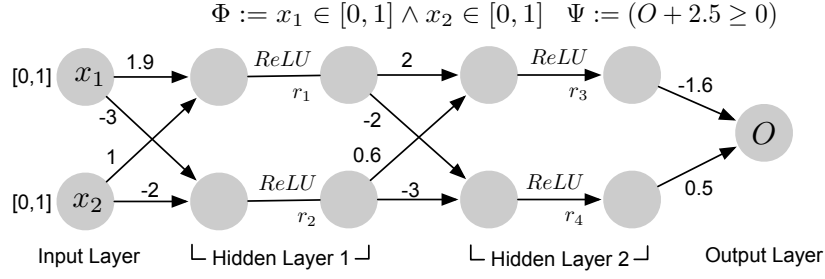
To exemplify the power of our order-guided exploration, we propose two variants of *Oliva*:

- *Oliva*^{GR} employs a greedy exploitation strategy, which always prioritizes the sub-problems that have a higher likelihood of containing counterexamples. This approach focuses on the most suspicious regions of the sub-problem space, and is likely to quickly expose the counterexamples such that the verification can be concluded quickly;
- To avoid overfitting of our proposed order, we also devise *Oliva*^{SA} inspired by simulated annealing [24], the famous stochastic optimization technique. The approach works similarly to simulated annealing: it maintains a variable called *temperature* that keeps decreasing throughout the process, which can control the trade-off between “exploitation” and “exploration”. At the initial stage, the temperature is high, and so the algorithm allows more chances of exploring the sub-problems that are not promising; as the temperature goes down, it converges to the optimal sub-problem. As the exploration of the search space has been done at the early stage of the algorithm, it is likely to converge to the globally optimal sub-problems and thereby find counterexamples. In this way, we mitigate the potential issue of being too greedy, and aim to strike a balance between “exploitation” and “exploration” in the search for suspicious sub-problems.

Evaluation

To evaluate the performance of *Oliva*, we perform a large scale of experiments on 690 problem instances spanning over 5 neural network models associated with the commonly-used datasets MNIST and CIFAR-10. By a comparison with the state-of-the-art verification approaches, we demonstrate the speedup of *Oliva*, for up to 25× in MNIST, and up to 80× in CIFAR-10. Moreover, we also show the breakdown results of *Oliva* for problem instances that are finally certified and the instances that are finally falsified. Experimental results show that *Oliva* is particularly efficient for those instances that are falsified, which demonstrates the effectiveness of our approach.

For a verification problem whose result is unknown beforehand, it is always desired to reach the conclusion as quickly as possible. Given that verification of neural networks is typically time- and resource-consuming, our approach provides a meaningful way to accelerate



■ **Figure 1** A neural network N and specification.

the verification process. While performing verification of neural networks with the aim of finding counterexamples sounds similar to approaches like testing or adversarial attacks, our approach differs fundamentally from those approaches, in the sense that, while those approaches deal with a single input each time and so they can never exhaust the search space, our approach deals with sub-problems that are finitely many, and so we can finally provide rigorous guarantees for specification satisfaction of neural networks.

Paper organization

The rest of the paper is organized as follows: §2 overviews our approach by using an illustrative example; §3 introduces the necessary technical background; §4 presents the proposed incremental verification approach; §5 presents our experimental evaluation results; §6 discusses related work. Conclusion and future work are presented in §7.

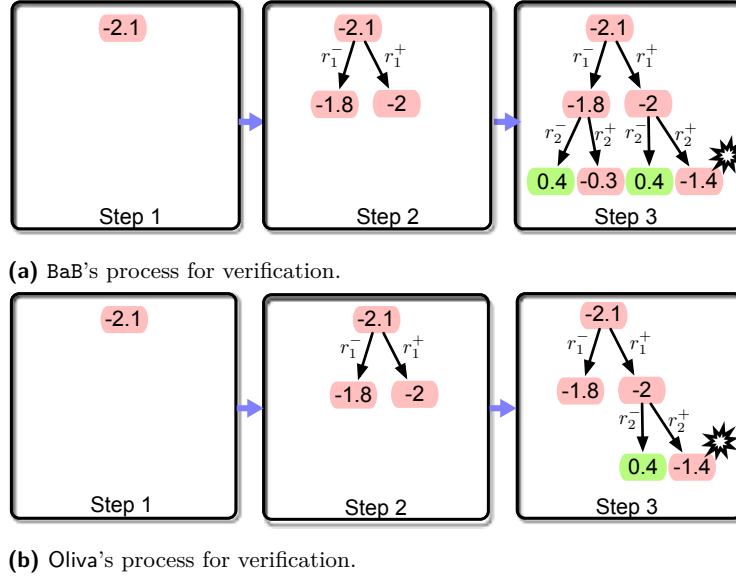
2 Overview of The Proposed Approach

In this section, we use an example to illustrate how the proposed approach solves neural network verification problems.

2.1 Verification Problem and BaB Approach

Fig. 1 depicts a (feed-forward) neural network N . It has an input layer, an output layer, and two *hidden layers* that are *fully-connected*, namely, the output of each hidden layer is computed by taking the weighted sum of the output of the previous layer, and applying the ReLU activation function. The output O of the neural network is computed by taking only the weighted sum of the output of the second hidden layer (without activation function). The weights of each layer are as labeled in Fig. 1. This neural network N is expected to satisfy such a specification: for any input $(x_1, x_2) \in [0, 1] \times [0, 1]$, it should hold that the output $O + 2.5 \geq 0$. Verification aims to give a formal proof to certify that N indeed satisfies the specification, or give a counterexample instead if N does not satisfy the specification.

As the state-of-the-art neural network verification approach, Branch-and-Bound (BaB) [5] has overarched several famous verification tools, such as $\alpha\beta$ -Crown [51]. The application of BaB to neural network verification often relies on the combination with off-the-shelf verifiers, and a common choice involves the family of approximated verifiers. These verifiers can efficiently decide whether the neural network satisfies the specification, by constructing a convex over-approximation of neural network outputs: if the over-approximated output satisfies the specification, the original output must also satisfy; however, if not, it does not indicate that the original output also violates the specification, and so it may raise a *false alarm* that reports a specification violation that is actually not existent.



■ **Figure 2** Neural network verification problem and solution via BaB (BaB *vs.* Oliva).

BaB is introduced to solve this problem. It is essentially a “divide-and-conquer” strategy that splits the problem when necessary and applies approximated verifiers to sub-problems to reduce the occurrences of false alarms. In this context, a problem can be split by predicating over the sign of the input of a ReLU function (i.e., by allowing the ReLU’s input to be positive or negative only), such that it can be decomposed to two linear functions each of which is easier to handle. BaB decides whether a (sub-)problem is needed to be split further, based on a value \hat{p} returned from the approximated verifier. Intuitively, \hat{p} indicates *how far* the specification is from being violated by the over-approximation of a (sub-)problem. If \hat{p} is positive, it implies that problem has been verified and so there is no need to split it; otherwise, it implies that the problem cannot be verified, and so BaB needs to check whether it is a false alarm, by checking whether the counterexample reported by the verifier is a spurious one. Fig. 2a illustrates how BaB solves the verification problem in Fig. 1:

- Step 1** – BaB first applies a verifier to the original verification problem, which returns a negative $\hat{p} = -2.1$. By validating the counterexample reported by the verifier, BaB identifies that it is a false alarm and decides to split the problem;
- Step 2** – BaB splits the problem identified by the root node, and applies the verifier to the two sub-problems respectively. Again, it identifies that the verifier raises a false alarm for each of the sub-problems. Therefore, it needs to split both sub-problems;
- Step 3** – Similarly, BaB applies the verifier in turn to the newly expanded sub-problems. It manages to verify two sub-problems (i.e., the two nodes both with $\hat{p} = 0.4$), and also identifies a false alarm (i.e., the node with $\hat{p} = -0.3$). In the node with $\hat{p} = -1.4$, BaB finds that the counterexample associated with this sub-problem is a real one; because having one such real counterexample suffices to show that the specification can be violated, BaB terminates the verification at this point with a verdict that the specification is violated.

2.2 The Proposed Approach

For a verification problem that is considerably difficult, BaB can often produce a space that contains a large number of sub-problems. However, as shown in Fig. 2a, the classic BaB explores these sub-problems by a naive “first come, first served” order, which ignores the difference of the sub-problems in terms of their importance, and thus can be very inefficient.

To bridge this gap, we propose an approach that explores the space of sub-problems guided by an order of importance of different sub-problems. Specifically, we identify the importance of different sub-problems by their likelihood of containing counterexamples, in the sense that the more likely a sub-problem contains counterexamples, it should be more prioritized in the exploration of the sub-problem space – once we can find a counterexample there, we can immediately terminate the verification and draw a conclusion. Even if it does not manage to find a counterexample in any problem, it just visits the sub-problems in a different order from the original BaB, so it still would not be much slower.

Fig. 2b demonstrates how our approach solves the same verification problem in Fig. 1.

- In early stages, Oliva works the same as BaB, i.e., it visits the (sub-)problems and obtains the same feedback \hat{p} from the verifier, and based on that, it decides to split the (sub-)problems and apply verifiers to sub-problems;
- The difference comes from the rightmost plot of Fig. 2b, in which Oliva prioritizes the sub-problem that has $\hat{p} = -2$ rather than its sibling that has $\hat{p} = -1.8$ to expand. This is because \hat{p} is a indicator that reflects how far the sub-problem is from being violated (based on the over-approximation performed by the verifier), and in this case $\hat{p} = -2$ signifies that this sub-problem has been violated *more* than the other and so it is more likely to find a counterexample there. Indeed, Oliva manages to find a real counterexample in one of its sub-problems, thereby terminating the verification immediately after that.

By this strategy, we manage to save two visits to sub-problems compared to BaB, each of which consists of an expensive problem solving process, and therefore, it reaches the conclusion of the problem more efficiently than BaB.

A variant of our approach inspired by simulated annealing

Essentially, our proposed approach changes the naive “breadth-first” exploration of the sub-problem space in the classic BaB to an intelligent way guided by the likelihood of finding counterexamples in different sub-problems. However, as this guidance consists of a total order over different branches, it may lose some chances of finding counterexamples in some branches that initially seem not promising.

To mitigate this issue, we also propose a variant of our approach, inspired by simulated annealing [24]. It works as follows: throughout the process, we maintain a temperature that keeps decreasing slowly, which is used to control the trade-off between “exploitation” and “exploration”. Initially, when the temperature is high, our algorithm tends to explore the sub-problem space by assigning a considerably high probability to accept a sub-problem that is not promising; as temperature goes down, it converges to the optimal sub-problem; due to the exploration of the search space at the initial stage, the algorithm is likely to converge to the globally optimal sub-problems. In this way, we can mitigate the side-effect of being too greedy, and strike a balance between “exploitation” and “exploration”.

3 Preliminaries

In this section, we first introduce the neural network verification problem, and then introduce the state-of-the-art verification approach called *branch-and-bound* (BaB).

3.1 Neural Network Verification Problem

In this paper, we consider feed-forward neural networks, as depicted in Fig. 1.

► **Definition 1** (Neural networks). A (feed-forward) neural network $N : \mathbb{R}^n \rightarrow \mathbb{R}^m$ maps an n -dim input to an m -dim output (see Fig. 1 for an example). It accomplishes the mapping by alternating between affine transformations (parametrized by weight matrix W_i and bias vector B_i) and non-linear activation functions σ ; namely, the output \mathbf{x}_i of layer i ($i \in \{1, \dots, L\}$) is computed based on the output \mathbf{x}_{i-1} of layer $i-1$, as follows: $\mathbf{x}_i = \sigma(W_i \mathbf{x}_{i-1} + B_i)$. Here, the computation unit that computes each dimension of \mathbf{x}_i is called a neuron; in each layer i , the number of neurons is equivalent to the number of dimensions of \mathbf{x}_i . Specially, \mathbf{x}_0 is the input and \mathbf{x}_L is the output of the neural network. There could be different choices for the non-linear activation functions, such as ReLU, sigmoid and tanh; following many existing works [25], we adopt Rectified Linear Unit (ReLU) (i.e., $\text{ReLU}(x) = \max(0, x)$) as the activation function in our neural networks.

Specifications are logical expressions that formalize users' desired properties about neural networks. In this paper, we adopt the following notation in Def. 2 as our specification formalism, and later we show that it can be used to formalize commonly-used properties, such as *local robustness* of neural networks.

► **Definition 2** (Specification). We denote by a pair (Φ, Ψ) a specification for neural networks, where Φ is called an input specification that predicates over the input region, and Ψ is called an output specification that predicates over the output of a neural network. Specifically, we denote the output specification Ψ as follows: $f(N(\mathbf{x}_0)) > 0$, where $f : \mathbb{R}^m \rightarrow \mathbb{R}$ is a function that maps an m -dim vector to a real number.

► **Definition 3** (Verification problem). A verification problem aims to answer the following question: given a neural network N and a specification (Φ, Ψ) , whether $\Psi(N(\mathbf{x}))$ holds, for any input \mathbf{x} that holds $\Phi(\mathbf{x})$. A verifier is a tool used to answer a verification problem: it either returns **true**, certifying N 's satisfaction to the specification, or returns **false** with a counterexample $\hat{\mathbf{x}}$, which is an input that holds $\Phi(\hat{\mathbf{x}})$ but does not hold $\Psi(\hat{\mathbf{x}})$.

We now explain how our notations can be used to formalize a neural network verification problem against local robustness, which is a property often considered in the domain of image classification. Local robustness requires that a neural network classifier should make consistent classification for two images \mathbf{x} and \mathbf{x}_0 , where \mathbf{x} is produced by adding small perturbations to \mathbf{x}_0 . Formally, given a reference input \mathbf{x}_0 , $\Phi(\mathbf{x})$ requires that the input \mathbf{x} must stay in the region $\{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_0\|_\infty \leq \epsilon\}$, where $\|\cdot\|_\infty$ denotes the ℓ^∞ -norm distance metric and ϵ is a small real value, and $\Psi(N(\mathbf{x}))$ requires that $\min_{1 \leq i \leq m, i \neq i_l} (N(\mathbf{x})_{i_l} - N(\mathbf{x})_i) > 0$, where $N(\mathbf{x})_i$ denotes the i -th component of the output vector $N(\mathbf{x})$, and i_l is the label of \mathbf{x}_0 inferred by the neural network, i.e., $i_l = \arg \max_{1 \leq i \leq m} N(\mathbf{x}_0)_i$.

3.2 Branch-and-Bound (BaB) – State-of-the-Art Verification Approach

There have been various approaches proposed to solve the neural network verification problem. Exact encoding [23, 47, 7] formalizes the inference process and the specification of a neural network to be an SMT or optimization problem and solves them by dedicated [23] or off-the-shelf [47, 7] solvers. These approaches are sound and complete, but due to the non-linearity of neural network inferences, they suffer from severe scalability issues and cannot handle neural networks of large sizes. To resolve this issue, approximated approaches [42, 41, 44] over-approximate the output of neural networks by linear relaxation, and they can certify the satisfaction of specification if the over-approximated output satisfies the specification. These approaches are typically efficient and sound; however, they are not complete, i.e., they

may raise false alarms with spurious counterexamples, if the over-approximated output does not satisfy the specification. As the state-of-the-art, Branch-and-Bound (BaB) [5] employs approximated verifiers and splits the problem when necessary (i.e., if a false alarm arises in a problem), because by solving sub-problems it can improve the precision of approximation and reduce the occurrence of false alarms. BaB keeps problem splitting until all of the sub-problems are certified (such that the original problem can be certified) or a real counterexample is detected with a sub-problem (such that the original problem is falsified).

In the following, we explain the necessary ingredients and detailed workflow of BaB.

Approximated verifiers

Approximated verifiers, denoted as **LpVerifier**, are a class of verifiers that solve a verification problem by computing an over-approximation of the output region of neural networks – if the over-approximation satisfies the specification, it implies that the original output region must also satisfy it. Typically, **LpVerifier** can return a tuple $\langle \hat{p}, \hat{\mathbf{x}} \rangle$, where $\hat{p} \in \mathbb{R}$ is called a *verifier assessment*, which is a quantity that indicates the *extent* to which the over-approximated output satisfies the specification. Formally, given an **LpVerifier**, it can construct a region $\hat{\Omega} \supseteq \{N(\mathbf{x}) \mid \Phi(\mathbf{x}) = \text{true}\}$, which over-approximates the output region of a neural network N ; then, **LpVerifier** computes \hat{p} as follows: $\hat{p} := \min_{\mathbf{y} \in \hat{\Omega}} f(\mathbf{y})$, where f is as defined in Def. 2. If \hat{p} is positive, it implies that the original output also satisfies the specification, and so it can certify the satisfaction of neural networks. Conversely, if \hat{p} is negative, **LpVerifier** deems that the specification is violated and provides a counterexample input $\hat{\mathbf{x}}$ as an evidence. However, due to the over-approximation, this reported violation of **LpVerifier** could be a false alarm and the counterexample $\hat{\mathbf{x}}$ may be a spurious one, i.e., the corresponding output $N(\hat{\mathbf{x}})$ of $\hat{\mathbf{x}}$ actually satisfies the specification. In this case, just based on the result of **LpVerifier**, we cannot infer whether N holds or violates the specification. This situation is referred to as the *completeness issue* of approximated verifiers.

There have been various approximation strategies that can be used to implement approximated verifiers, e.g., *DeepPoly* [44] and *ReluVal* [50], and mostly, the over-approximation is accomplished by using linear constraints to bound the output range of the non-linear ReLU functions. In particular, we assume that our adopted approximated verifiers hold the *monotonicity* property, namely, for two output regions of neural networks that hold $\Omega_1 \subset \Omega_2$, if we over-approximate them by using the same approximated verifier **LpVerifier**, the obtained over-approximation $\hat{\Omega}_1$ and $\hat{\Omega}_2$ hold that $\hat{\Omega}_1 \subseteq \hat{\Omega}_2$. While there can be different options of over-approximation strategies, our proposed approach is orthogonal to them, so we can adopt any approximated verifiers that hold our assumption about monotonicity.

Branch-and-Bound (BaB)

BaB is the state-of-the-art neural network verification approach, and has been adopted in several advanced verification tools, such as $\alpha\beta$ -Crown [51] and Marabou [52]. It is essentially a “divide-and-conquer” strategy, that divides a verification problem adaptively and applies off-the-shelf verifiers (such as approximated verifiers) to solve the sub-problems. Because approximated verifiers often achieve better precision on sub-problems, BaB can thus overcome the weaknesses of the plain application of approximated verifiers to the original problem and resolve the issues of false alarms.

Before looking into the details of BaB, we first introduce ReLU specification, which is an important notion in the algorithm of BaB.

■ **Algorithm 1** Branch-and-Bound (BaB) [5].

Require: A neural network N , an input specification Φ , an output specification Ψ , an approximated verifier $\text{LpVerifier}(\cdot)$, and a ReLU selection heuristic $H(\cdot)$.

Ensure: A *verdict* $\in \{\text{true}, \text{false}\}$.

```

1:  $Q \leftarrow \{\top\}$ 
2:  $\mathcal{T} \leftarrow \emptyset$ 
3: function BAB( $N, \Phi, \Psi, Q$ )
4:   if EMPTY( $Q$ ) then
5:     return true
6:    $\Gamma \leftarrow \text{POP}(Q)$ 
7:    $\langle \hat{p}, \hat{x} \rangle \leftarrow \text{LpVerifier}(N, \Phi, \Psi, \Gamma)$ 
8:    $\mathcal{T} \leftarrow \mathcal{T} \cup \{\langle \Gamma, \hat{p} \rangle\}$ 
9:   if  $\hat{p} < 0$  then
10:    if VALID( $\hat{x}, N, \Psi$ ) then
11:      return false
12:    else
13:       $r_k \leftarrow H(\Gamma)$ 
14:      for  $a \in \{r_k^+, r_k^-\}$  do
15:        PUSH( $Q, \Gamma \wedge a$ )
16:   return BAB( $N, \Phi, \Psi, Q$ )

```

► **Definition 4** (ReLU specification). Let N be a neural network consisting of K neurons, and $\hat{x}_i \in \mathbb{R}$ ($i \in \{1, \dots, K\}$) be the input for the ReLU function in the i -th neuron. An atomic proposition AP w.r.t. the i -th neuron is defined as either $\hat{x}_i \geq 0$ (written as r_i^+) or $\hat{x}_i < 0$ (written as r_i^-). Then, a ReLU specification Γ is defined as the conjunction of a sequence $\text{Set}(\Gamma) := \{\text{AP}_1, \dots, \text{AP}_{|\Gamma|}\}$ of atomic propositions, where each $\text{AP} \in \text{Set}(\Gamma)$ is defined w.r.t. a distinct neuron. $|\Gamma|$ is the number of atomic propositions in $\text{Set}(\Gamma)$; specially if $|\Gamma| = 0$, Γ is denoted as \top . Moreover, we define a refinement relation \prec over ReLU specifications, namely, given two ReLU specifications Γ_i and Γ_j , we say Γ_j refines Γ_i (denoted as $\Gamma_i \prec \Gamma_j$) if and only if $\text{Set}(\Gamma_i) \subset \text{Set}(\Gamma_j)$.

By selecting a neuron i , we can split the ReLU function into two linear functions, each identified by a predicate r_i^+ or r_i^- over the input of ReLU. Thereby, a neural network verification problem boils down to two sub-problems, for each of which LpVerifier does not need to perform over-approximation for the ReLU function in the i -th neuron.

The workflow of BaB is presented in Alg. 1. In Alg. 1, we allow LpVerifier to take an additional argument, namely, a ReLU specification Γ , which identifies a sub-problem by adding the constraints in $\text{Set}(\Gamma)$ to constrain the inputs of a number of selected ReLU functions. It uses a FIFO queue Q to maintain the problem to be solved, which is initialized to be a set that consists of \top only, identifying the original verification problem.

- i) First, it applies LpVerifier to the original problem (Line 7): if LpVerifier returns a positive \hat{p} , or a negative \hat{p} with a valid counterexample \hat{x} (Line 10), then verification can be terminated with a verdict returned accordingly;
- ii) In the case it returns a negative \hat{p} with a spurious counterexample (Line 12), it divides the verification problem into two sub-problems. This is achieved by first selecting a neuron (i.e., a ReLU) in the network according to a pre-defined ReLU selection heuristics H (Line 13), and then identifying two sub-problems each identified by an additional constraint on the input of the selected ReLU function (Line 15).

- iii) It applies `LpVerifier` to the new sub-problems respectively, and decides whether it needs to further split the sub-problem, following the same rule in Step ii. In Alg. 1, this is implemented by recursive call of the `BaB` function in Line 16.

In `BaB`, the ReLU selection heuristics H involves an order over different neurons such that it can select the next ReLU based on an existing ReLU specification. There has been a rich body of literature that proposes different ReLU selection strategies, such as `DeepSplit` [21] and `FSB` [10]. However, our proposed approach in §4 is orthogonal to these strategies and so it can work with existing strategies. In this work, we follow an existing neural network verification approach [48] and we adopt the state-of-the-art ReLU selection strategy in [21].

As a result of Alg. 1, it forms a binary tree \mathcal{T} (see Line 2 and Line 8) that records the history of problem splitting during the `BaB` process. In this tree, each node $\langle \Gamma, \hat{p} \rangle$ denotes a sub-problem, identified by a ReLU specification Γ and the verifier assessment \hat{p} returned by `LpVerifier` that signifies the satisfaction of the sub-problem.

► **Lemma 5** (Soundness and completeness). *The `BaB` algorithm is sound and complete.*

Proof. Soundness requires that if `BaB` returns `true`, the neural network must satisfy the specification. The soundness of `BaB` relies on that: 1) `LpVerifier` is sound; 2) the ReLU specifications identified by the leaf nodes of the `BaB` tree cover all the cases about the input conditions of the split ReLUs in the neural network.

Completeness requires that if `BaB` returns `false`, the neural network must violate the specification. The completeness of `BaB` relies on the fact that the counterexamples \hat{x} reported by `BaB` are all validated and so they must be real. Moreover, in the worst case if all neurons are split, the sub-problem will be linear and so a \hat{x} must be a real one if it exists.

Finally, `BaB` is guaranteed to return either `true` or `false` within a finite time budget, because the number of neurons in a neural network is finite. ◀

4 Oliva: The Proposed Verification Approach

As a “divide-and-conquer” strategy, `BaB` can produce a huge space that consists of quantities of sub-problems; however, as shown in Alg. 1, the existing `BaB` approach explores this space in a naive “first come, first served” manner (implemented by the *first-in-first-out* queue in Alg. 1 that stores the (sub)-problems to be solved), which can be very inefficient to exhaust the sub-problem space.

Our proposed approach involves exploring the `BaB` tree in an intelligent fashion, guided by the severity of different tree nodes. In §4.1, we showcase such a severity order, defined by the probability of finding counterexamples with a sub-problem. The intuition behind this order is that, by prioritizing sub-problems that are more likely to find counterexamples, we may quickly find a counterexample and thereby immediately terminate the verification. If we cannot find it after visiting all sub-problems, we achieve certification of the problem.

4.1 Counterexample Potentiality Order

We introduce an order over different sub-problems based on their probability of containing counterexamples. Given a node in `BaB` tree that identifies a sub-problem, we can infer this probability using the following two attributes:

Node depth. In a `BaB` tree, the depth of a node signifies the levels of problem splitting, and for more finely-split sub-problems, `LpVerifier` introduces less over-approximation. Because of this, if a node Γ with a greater depth is still deemed by `LpVerifier` as violating the specification (i.e., the verifier assessment \hat{p} is negative), it is more likely that Γ indeed contains real counterexamples.

Verifier assessment. Given a node in BaB tree that identifies a sub-problem, its \hat{p} returned by `LpVerifier` can be considered as a quantitative indicator of *how far* the sub-problem is from being violated. Due to our assumption about the monotonicity of `LpVerifier` in performing over-approximation, given a fixed `LpVerifier`, \hat{p} is correlated to the original output region of the neural network. Therefore, in the case \hat{p} is negative, the greater $|\hat{p}|$ is, there is a higher possibility that the original output region is closer to violation of the specification, and so it is more likely that the sub-problem contains a real counterexample.

Based on the above two node attributes, we define the suspiciousness of a BaB tree node, and then define a Counterexample POtentiality (CePO) order over different nodes.

► **Definition 6** (Suspiciousness of sub-problems). *Let Γ be a node of the BaB tree that has a verifier assessment \hat{p} (with a counterexample \hat{x} if $\hat{p} < 0$). The suspiciousness $\text{susp}(\Gamma) \in [0, 1] \cup \{+\infty, -\infty\}$ of the node Γ maps Γ to a real number as follows:*

$$\text{susp}(\Gamma) := \begin{cases} -\infty & \text{if } \hat{p} > 0 \\ +\infty & \text{if } \hat{p} < 0 \text{ and } \text{valid}(\hat{x}) \\ \lambda \frac{\text{depth}(\Gamma)}{K} + (1 - \lambda) \frac{\hat{p}}{\hat{p}_{\min}} & \text{otherwise} \end{cases}$$

where $\lambda \in [0, 1]$ is a parameter that controls the weights of the two attributes, and K is the total number of neurons (i.e., ReLUs) in the network.

Intuitively, the suspiciousness of a node encompasses a heuristic that estimates the probability of the relevant sub-problem violating the specification. It is particularly meaningful in the case when $\hat{p} < 0$ and the counterexample is spurious, and set to be $+\infty$ if the sub-problem is provably violated, and $-\infty$ if the sub-problem is certified. By their suspiciousness, we define the CePO order over different nodes as follows:

► **Definition 7** (Counterexample potentiality (CePO) order). *Let Γ_1 and Γ_2 be two nodes in a BaB tree, and \hat{p}_1 and \hat{p}_2 be verifier assessments for Γ_1 and Γ_2 , respectively. We define a CePO order \sqsubset between the two nodes as follows:*

$$\Gamma_1 \sqsubset \Gamma_2 \quad \text{iff} \quad \text{susp}(\Gamma_1) < \text{susp}(\Gamma_2)$$

The CePO order allows us to sort the nodes in BaB tree,¹ and so it can serve as a guidance to our verification approach in §4.2 and §4.3.

4.2 Oliva^{GR}: Greedy Exploration of BaB Tree

The Oliva^{GR} algorithm, presented in Alg. 2, implements a greedy strategy that explores the space of BaB tree nodes guided by the CePO order. Compared to the classic BaB, it always selects the most suspicious node to expand, such that it can maximize the probability of finding counterexamples and thereby conclude the verification problem efficiently.

The algorithm begins with applying the approximated verifier `LpVerifier` to the original verification problem, identified by the ReLU specification ε (Line 2), which returns a tuple $\langle \hat{p}, \hat{x} \rangle$ containing a verifier assessment \hat{p} , possibly followed by a counterexample \hat{x} . The suspiciousness $\text{susp}(\varepsilon)$ of the root node is then computed based on this assessment. At this point, if $\hat{p} > 0$ or if $\hat{p} < 0$ with a valid counterexample \hat{x} , the algorithm can be immediately

¹ In the case if two nodes have the same suspiciousness, we simply impose a random order; in the case both are $-\infty$ or $+\infty$, the comparison is meaningless and so we do not need to sort them.

■ **Algorithm 2** Oliva^{GR}: The proposed greedy algorithm.

Require: A neural network N , input and output specification Φ and Ψ , an approximated verifier $\text{LpVerifier}(\cdot)$, a ReLU selection heuristic $H(\cdot)$, and a hyperparameter λ .

Ensure: A *verdict* $\in \{\text{true}, \text{false}, \text{timeout}\}$

```

1:  $\mathcal{T} \leftarrow \{\varepsilon\}$ 
2:  $\langle \hat{p}, \hat{x} \rangle \leftarrow \text{LpVerifier}(N, \Phi, \Psi, \varepsilon)$ 
3:  $R(\varepsilon) \leftarrow \text{susp}(\varepsilon)$  ▷ a metric for child selection
4: if  $\hat{p} < 0$  and not  $\text{valid}(\hat{x})$  then
5:   while not reach termination condition do
6:      $\text{GREEDYBAB}(\varepsilon, N, \Phi, \Psi)$ 
7:   return  $\begin{cases} \text{true} & \text{if } R(\varepsilon) = -\infty \\ \text{false} & \text{if } R(\varepsilon) = +\infty \\ \text{timeout} & \text{otherwise} \end{cases}$ 
8: else
9:   return  $\begin{cases} \text{true} & \text{if } \hat{p} > 0 \\ \text{false} & \text{if } \hat{p} < 0 \text{ and } \text{valid}(\hat{x}) \end{cases}$ 

10: function  $\text{GREEDYBAB}(\Gamma, N, \Phi, \Psi)$ 
11:    $r_k \leftarrow H(\Gamma)$ 
12:   if  $\Gamma \cdot r_k^+ \in \mathcal{T}$  then ▷ check existence of children
13:      $\Gamma^* \leftarrow \Gamma \cdot a^* \quad \text{s.t. } a^* \leftarrow \arg \max_{a \in \{r_k^+, r_k^-\}} R(\Gamma \cdot a)$ 
14:      $\text{GREEDYBAB}(\Gamma^*, N, \Phi, \Psi)$ 
15:   else
16:     for  $a \in \{r_k^+, r_k^-\}$  do ▷  $\mathcal{T}$  expansion via BaB.
17:        $\langle \hat{p}, \hat{x} \rangle \leftarrow \text{LpVerifier}(N, \Phi, \Psi, \Gamma \cdot a)$  ▷ apply  $\text{LpVerifier}$  with  $\Gamma \wedge a$ 
18:        $R(\Gamma \cdot a) \leftarrow \text{susp}(\Gamma \cdot a)$  ▷ compute  $R$ 
19:        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\Gamma \cdot a\}$  ▷ add to the tree of  $\Gamma \wedge a$ 
20:    $R(\Gamma) \leftarrow \arg \max_{a \in \{r_k^+, r_k^-\}} R(\Gamma \cdot a)$ 

```

terminated with a conclusive verdict (Line 9). Otherwise, the algorithm enters its main loop where it iteratively calls GREEDYBAB to split and verify the sub-problems until a termination condition is reached (Line 5-7). We elaborate on the termination condition later.

The GREEDYBAB function implements the main process of tree exploration and problem splitting. By each of its execution, it selects the maximal sub-problem in terms of the CePO order in the tree and applies LpVerifier to its subsequent sub-problems with the aim of finding counterexamples. The function begins with selecting a ReLU r_k as a successor of the current node (Line 11) using the pre-defined ReLU selection heuristics H (see §3.2), and then it recursively calls GREEDYBAB until it reaches the greatest node Γ , in terms of the CePO order, whose children are not expanded yet (Line 12-14). After reaching such a node Γ , it expands the children of Γ , by applying LpVerifier respectively to the two sub-problems identified by the children of Γ (Line 16-17). It also computes and records the suspiciousness of the newly expanded children (Line 18), and updates \mathcal{T} that keeps track of the tree (Line 19). Lastly, the greater suspiciousness over the children are propagated backwards to ancestor nodes until the root (Line 20), serving as a reference for future node selection.

Termination condition

The main loop in Line 5 of Alg. 2 can be terminated on the satisfaction of any of the following three conditions:

- $R(\varepsilon) = -\infty$: This implies that all leaf nodes have been verified successfully (i.e., $\hat{p} > 0$ for all leaves, otherwise by Line 20 $R(\varepsilon)$ cannot be $-\infty$), allowing the algorithm to return a true verdict that certifies the specification satisfaction;
- $R(\varepsilon) = +\infty$: This implies that a valid counterexample has been found in some leaf node, and so by Line 20 its suspiciousness of $+\infty$ can be back-propagated to the root node; in this case, the algorithm can be terminated with a false verdict;
- **timeout**: This occurs when the algorithm exceeds its allocated time budget without reaching either of the above conclusive verdicts.

The three termination conditions correspond to the three cases of return in Line 7.

4.3 Oliva^{SA}: Simulated-Annealing-Style Exploration of BaB Tree

The greedy strategy in §4.2 exploits the CePO order in the exploration of BaB tree, so it can efficiently move towards the sub-problems that are more likely to find counterexamples. However, as it is a greedy strategy, in the case if the CePO order is not sufficiently precise, the verification process can be easily trapped into a local optimum, and miss the chances of finding counterexamples in other branches than the one suggested by the CePO order.

To bridge this gap, we propose an approach that adapts the classic framework of *simulated annealing*, which not only follows the CePO order, but also takes other branches into account. As a consequence, during the verification, it strikes a balance between “exploitation” of the suspicious branches and “exploration” of the less suspicious branches.

“Hill climbing” vs. “Simulated annealing”

Hill climbing is a stochastic optimization technique that aims to find the optimum of a black-box function. Due to the black-box nature of the objective function, it relies on sampling in the search space and selects only the samples that optimize the objective function as the direction to move. Therefore, this is a greedy strategy, similarly to our proposed approach Oliva^{GR} in §4.2. While Oliva^{GR} does not need to sample the search space, it can select the sub-problem that is more promising, and thereby move towards the direction that is more likely to achieve the objective. As a consequence, they both suffer from the issue of “local optima”.

In the field of stochastic optimization, *simulated annealing* [24] is an effective approach to mitigate the issue of “local optima” in hill climbing. It is inspired by the process of *annealing* in metallurgy, in which the temperature is initially high but slowly decreases such that the physical properties of metals can be stabilized. In simulated annealing, there is also a temperature that slowly decreases throughout the process: when the temperature is initially high, it tends to explore the search space and assigns a considerably high probability to accept a sample that is not the optimal; as temperature goes down, it converges to the optimum that is likely to be the global one, thanks to the exploration of the search space at the initial stage.

Our proposed approach Oliva^{SA} adapts simulated annealing to our problem setting, and the core idea involves that, at the initial stage, we allow more chances of exploring the branches that are less promising. Then, after we have comprehensively explored the search space and obtained the information about the suspiciousness of different branches, we tend to exploit the branches that are more suspicious, namely, that are more likely to contain counterexamples.

■ **Algorithm 3** Oliva^{SA}: The proposed simulated annealing algorithm.

Require: A neural network N , input and output specification Φ and Ψ , an approximated verifier $\text{LpVerifier}(\cdot)$, a ReLU selection heuristic $H(\cdot)$, and a hyperparameter λ .

Ensure: A *verdict* $\in \{\text{true}, \text{false}, \text{timeout}\}$

```

1:  $\mathcal{T} \leftarrow \{\varepsilon\}$ 
2:  $\langle \hat{p}, \hat{x} \rangle \leftarrow \text{LpVerifier}(N, \Phi, \Psi, \varepsilon)$ 
3:  $R(\varepsilon) \leftarrow \text{susp}(\varepsilon)$  ▷ a metric for child selection
4:  $T \leftarrow T_{max}$  ▷ temperature
5: if  $\hat{p} < 0$  and not  $\text{valid}(\hat{x})$  then
6:   while not reach termination condition do
7:      $T \leftarrow \alpha \cdot T$  ▷  $T$  is decreased by  $(1 - \alpha)T$  in each iteration
8:      $\text{SABAB}(\varepsilon, N, \Phi, \Psi)$ 
9:   return  $\begin{cases} \text{true} & \text{if } R(\varepsilon) = -\infty \\ \text{false} & \text{if } R(\varepsilon) = +\infty \\ \text{timeout} & \text{otherwise} \end{cases}$ 
10: else
11:   return  $\begin{cases} \text{true} & \text{if } \hat{p} > 0 \\ \text{false} & \text{if } \hat{p} < 0 \text{ and } \text{valid}(\hat{x}) \end{cases}$ 

12: function  $\text{SABAB}(\Gamma, N, \Phi, \Psi)$ 
13:    $r_k \leftarrow H(\Gamma)$ 
14:   if  $\Gamma \cdot r_k^+ \in \mathcal{T}$  then
15:      $\Delta p \leftarrow \exp\left(\frac{\min R(\Gamma \cdot a) - \max R(\Gamma \cdot a)}{T}\right)$  s.t.  $a \in \{r_k^+, r_k^-\}$ 
16:      $\Gamma^* \leftarrow \Gamma \cdot a^*$  s.t.  $a^* \leftarrow \begin{cases} \text{randomly choose } r_k^+ \text{ or } r_k^- & \text{if } \text{rand}(0, 1) < \Delta p \\ \arg \max_{a \in \{r_k^+, r_k^-\}} R(\Gamma \cdot a) & \text{otherwise} \end{cases}$ 
17:      $\text{SABAB}(\Gamma^*, N, \Phi, \Psi)$ 
18:   else
19:     for  $a \in \{r_k^+, r_k^-\}$  do ▷  $\mathcal{T}$  expansion via BaB.
20:        $\langle \hat{p}, \hat{x} \rangle \leftarrow \text{LpVerifier}(N, \Phi, \Psi, \Gamma \cdot a)$  ▷ apply  $\text{LpVerifier}$  with  $\Gamma \wedge a$ 
21:        $R(\Gamma \cdot a) \leftarrow \text{susp}(\Gamma \cdot a)$  ▷ compute  $R$ 
22:        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\Gamma \cdot a\}$  ▷ add to the tree of  $\Gamma \wedge a$ 
23:    $R(\Gamma) \leftarrow \arg \max_{a \in \{r_k^+, r_k^-\}} R(\Gamma \cdot a)$ 

```

Algorithm details

The Oliva^{SA} algorithm is presented in Alg. 3. It also starts with checking the original verification problem (Line 2), and enters the loop of tree exploration if the original problem cannot be solved by LpVerifier (Line 5). However, compared to Oliva^{SA}, it has a notable difference about the adoption of a temperature T (Line 4) which is a global variable that keeps decreasing in each loop of the algorithm (Line 7).

In the function SABAB , the temperature T is used when selecting the nodes to proceed. In the case if the children of the current node Γ have been expanded (Line 14), unlike Oliva^{GR} that always prefers the most suspicious child, Oliva^{SA} selects a child according to the policy adapted from the original simulated annealing:

- It first computes an acceptance probability Δp (Line 15), by which it determines whether the selection of a child that is less promising is acceptable. This probability Δp is decided by both the difference of “energy” (defined by the suspiciousness difference between two children) and the temperature T ;
- In the original simulated annealing, a sample that is more promising can be accepted in any case, and a sample that is less promising can be accepted with the probability Δp . In our context, we adapt this policy as follows (Line 16):
 - If a random value in $[0, 1]$ is less than Δp , we randomly select a child from the two children of Γ with the same probability, despite the CePO order over the two children;
 - Otherwise, we select the more suspicious child following the CePO order.

The selected child will be used as the argument for the recursive call of SABAB, in order to proceed towards a node whose children are not expanded. After achieving such a node Γ , it expands the children of Γ (Lines 19-22) by applying **LpVerifier** to each child of Γ , recording their suspiciousness (Line 21) and updating the BaB tree (Line 22). Lastly, it back-propagates the greater suspiciousness over the children until the root (Line 23), similarly to Oliva^{SA} .

This gradual transition from exploration to exploitation helps Oliva^{SA} to avoid premature convergence while ensuring an eventual focus on the promising branches. The termination conditions remain the same as in Oliva^{GR} , checking for full verification ($R(\varepsilon) = -\infty$), counterexample discovery ($R(\varepsilon) = +\infty$), or timeout. However, by pursuing a higher coverage of the sub-problem space before being greedy, Oliva^{SA} increases the possibility of discovering counterexamples that might be missed by the purely greedy strategy of Oliva^{GR} .

► **Theorem 8.** *Both Oliva^{GR} and Oliva^{SA} are sound and complete.*

Proof. The proofs of soundness and completeness of our approaches are similar to that of BaB in Lemma 5, so we skip the details. Intuitively, our approaches only introduce an order of visiting different nodes in BaB tree, but hold all the conditions that are necessary for the soundness and completeness of BaB. ◀

5 Experimental Evaluation

5.1 Experiment Settings

Baselines and Metrics

We compare the two versions of Oliva, namely, Oliva^{GR} and Oliva^{SA} , with three state-of-the-art neural network verification tools $\alpha\beta\text{-Crown}$ [59, 51], **NeuralSAT** [11], and **BaB**-baseline, as presented in §3.2. The settings of baseline approaches are detailed below:

- **BaB**-baseline is implemented based on the ERAN framework [41, 44], which employs LP-based triangle relaxation for bounding the output of ReLU functions, and utilizes DeepSplit [21] as the ReLU selection heuristics for selecting ReLU function to split, i.e., H in Alg. 1, Alg. 2, and Alg. 3.
- $\alpha\beta\text{-Crown}$ [59, 51] is applied with its default branch-and-bound settings. Moreover, a balanced strategy kFSB from FSB [10] is selected as the ReLU selection heuristics for selecting the ReLU function to split.
- **NeuralSAT** [11] is applied with its default DPLL(T) framework and we select the stabilized optimization as the neuron stability heuristics for ReLU activation pattern search. Additionally, it is registered with a random attack [8, 56] to reject the easily detected violation instances at the early stage of verification processes.

■ **Table 1** Benchmark details for the evaluation of verification.

Model	Architecture	Dataset	#Activations	# Instances	#Images
MNIST _{L2}	2×256 linear	MNIST	512	100	70
MNIST _{L4}	4×256 linear	MNIST	1024	78	52
OVAL21 _{BASE}	2 Conv, 2 linear	CIFAR-10	3172	173	53
OVAL21 _{WIDE}	2 Conv, 2 linear	CIFAR-10	6244	196	40
OVAL21 _{DEEP}	4sub Conv, 2 linear	CIFAR-10	6756	143	53

For our approaches, the hyperparameters are set as follows: $\lambda = 0.5$ (see Def. 6), $T_{max} = 1$ (see Line 4 of Alg. 3) and $\alpha = 0.99$ (see Line 7 of Alg. 3); in RQ3, we study the impact of these hyperparameters on the performance of our approaches.

We apply each of the five approaches (including two proposed approaches and three baseline approaches) to each of the verification problems, and we adopt 1000 secs as our time budget, following the VNN-COMP competition [30]. If an approach manages to solve the problem (either verifies the problem or reports a real counterexample), we deem this run as a *solved* verification process, and record the time cost for reaching the verification conclusion. In particular, our *Oliva*^{SA} is a stochastic approach, namely, its performance is subject to randomness. To compare its performance with other approaches, in Table 2, Table 3 and Fig. 5, we only show the performance of one random run; in Fig. 6, we particularly study the influence of randomness to *Oliva*^{SA}, by repeating each verification process for 5 times.

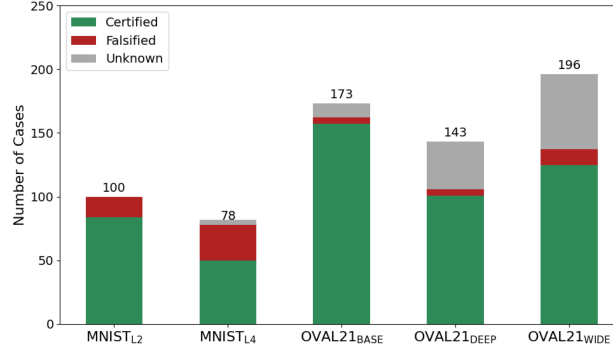
Our evaluation metrics include the number of instances solved by an approach and the time costs for each successful verification process. For comparison of two different approaches, we also compute the speedup rate, which is the ratio of the time costs of the two approaches.

Moreover, in order to understand whether our counterexample potentiality order works, we compare the performances of our tools with the baseline approach, respectively for the problems that are finally certified and the problems that are finally falsified. We further study the influence of the hyperparameters (including λ in Def. 6 and α that decides the change rate of temperature in Line 7 of Alg. 3) to the performances of our approaches. In the implementation of *Oliva*, we adopt the same approximated verifiers and ReLU selection heuristics as the BaB-baseline. The code and experimental data of *Oliva* are available online².

Datasets and Neural Networks

Our experimental evaluation uses two well-known datasets: MNIST, featuring images of handwritten digits for classification, and CIFAR-10, featuring images of various real-world objects like airplanes, cars, and animals, with networks tasked to identify each class. These datasets are standard benchmarks that have been widely used in the neural network verification community and adopted in the VNN-COMP [30], an annual competition in the community for comparing the performances of different verification tools. We evaluate two networks trained on MNIST that have fully-connected layers only, and three neural networks trained on CIFAR-10 that have both convolutional layer and fully-connected layers, with different network architectures, following common evaluation utilized in ERAN [32, 51] and OVAL21 [4, 5] benchmarks. More details of these benchmarks are presented in Table 1.

² <https://github.com/DeepLearningVerification/Oliva>



■ **Figure 3** The distribution of Certified/Falsified/Unknown cases generated by BaB-baseline.

Specifications

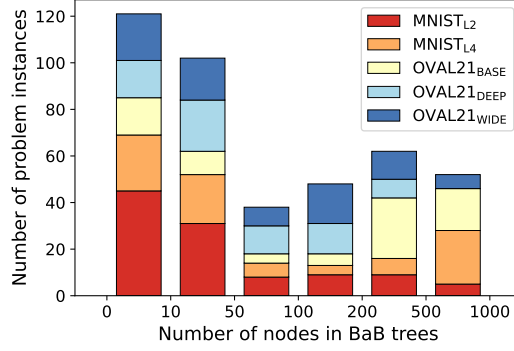
In our experiments, all our neural network models are used for image classification tasks, and all the specifications adopted are about local robustness of the neural network models. Each input specification defines an ϵ as the threshold for perturbation; and each output specification requires the expected label to be matched by the original input of the model. In total, we collected 690 problem instances. Table 1 under the column “# Instances” displays the number of instances in each model, and “# Images” shows the number of different images covered by the different instances. Fig. 3 demonstrates the distribution of verification verdicts across the five models in our benchmark set, according to our experimental results. The green portion denotes tasks eventually verified (“Certified”), the red portion denotes tasks that were shown to violate the property (“Falsified”), and the gray portion corresponds to tasks that remained inconclusive (“Unknown”).

In order to present a meaningful comparison, we need to avoid the verification problems that are too simple, for which the problem can be solved within a small number of problem splitting and so there is no need to expand the BaB tree too much. To that end, we perform a selection of parameters (i.e., ϵ in Def. 2) of input specifications, from a range of 0/255 to 16/255 under the L_∞ norm. Our approach involves a binary search-like algorithm to determine the proper perturbation values for each image, as follows:

1. Initially, we set 0/255 as the lower bound l and 16/255 as the upper bound u , and calculates the midpoint by taking $m = \frac{l+u}{2}$;
2. We apply BaB-baseline to the verification problem with m as the ϵ , and check the number of nodes in the BaB tree. If the tree size is greater than 1, we accept it as a candidate parameter and move to the next image; otherwise, based on the results of verification, we proceed to update m as like Step 1 and repeat Step 2 as follows:
 - If the specification is violated, it implies that ϵ is too large and so we set u to be m , and update m accordingly;
 - If the specification is satisfied, it implies that ϵ is too small and so we set l to be m , and update m accordingly.

This process continues until a pre-defined budget is exhausted.

Fig. 4 shows the distribution of node counts across all models for the verification instances. The distribution confirms that all instances involve multiple tree nodes, ensuring meaningful sub-problem selection. Notably, more than half of the tree sizes fall within the range of 100-1000 nodes, highlighting the importance of effective sub-problem selection. The complexity of most instances necessitates careful choice in verification instances.



■ **Figure 4** The distribution of the tree sizes generated by BaB-baseline.

Software and Hardware Setup

All experiments were conducted on an AWS EC2 instance running a Linux system with 16GB of memory and an 8-core Xeon E5 2.90 GHz CPU. All tools are developed in Python 3.9, and we used the GUROBI solver 9.1.2 [19] for the LP-based optimization. For each verification instance, we set a time budget as 1000 seconds which is consistent with VNN-COMP [30].

5.2 Evaluation

RQ1: Is Oliva more efficient than existing approaches?

In Table 2, we compare the performance of our approach with the state-of-the-art baseline approaches, BaB-baseline, $\alpha\beta$ -Crown and NeuralSAT, across all our verification problems.

We observe that, Oliva shows evident improvement for benchmarks including OVAL21_{BASE}, OVAL21_{DEEP} and OVAL21_{WIDE}. On the OVAL21_{BASE}, Oliva^{SA} solves 159 instances in an average of 155.29 seconds, far surpassing BaB-baseline (42 instances in 770.7 seconds) and $\alpha\beta$ -Crown (58 instances in 641.7 seconds). For OVAL21_{DEEP}, Oliva^{GR} solves 92 instances, compared to 33 by BaB-baseline and 55 by $\alpha\beta$ -Crown, while using less time (250.72 seconds compared to 552.24 seconds). The difference is even more pronounced for OVAL21_{WIDE}, where Oliva^{GR} solves 131 instances in an average of 240.16 seconds, greatly exceeding both BaB-baseline (40 instances in 733.73 seconds) and $\alpha\beta$ -Crown (63 instances in 557.51 seconds).

For MNIST_{L2} benchmarks that have simpler architectures compared to OVAL21, BaB-baseline performs well with 96 instances solved, while Oliva^{GR} achieves 99 solved instances with a faster average time (57.76 seconds). As the size of the network increases to MNIST_{L4}, $\alpha\beta$ -Crown declines to 44 instances, while Oliva^{GR} achieves 67 solved instances with an average time of 146.31 seconds. These results highlight the effectiveness and efficiency of Oliva, compared to the baseline approaches.

Table 3 presents a pairwise comparison, in which each cell indicates the number of problem instances solved by the approach listed in the row, but not solved by the approach listed in the column. Namely, we can perform pairwise comparison between two approaches by comparing the values in a pair of cells symmetric to the diagonal of Table 3. Compared to baseline approaches, our approaches demonstrate superior performances, as indicated by the green area that shows the number of additional problems solved by our approaches by not by baseline approaches; in comparison, the numbers in the red area, that includes the number of problems solved by baseline approaches, but not solved by our approaches, are much less. By comparing the performances between Oliva^{GR} and Oliva^{SA}, we find that Oliva^{GR} slightly

■ **Table 2** RQ1 – Overall comparison of different verification approaches, in terms of the number of solved problem instances and the time costs (in secs).

Model	BaB-baseline		$\alpha\beta$ -Crown		Neuralsat		Oliva ^{GR}		Oliva ^{SA}	
	Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	Time
MNIST _{L2}	96	126.41	87	51.32	99	32.37	95	96.79	99	57.76
MNIST _{L4}	65	194.74	44	428.03	54	392.04	67	146.31	53	142.72
OVAL21 _{BASE}	42	770.7	58	641.7	70	621.21	154	184.96	159	155.29
OVAL21 _{DEEP}	33	694.74	55	552.24	55	539.59	92	250.72	87	261.68
OVAL21 _{WIDE}	40	733.73	63	557.51	65	533.01	131	240.16	112	288.0

■ **Table 3** RQ1 – Pairwise comparison on the number of additional solved problem instances from all verification tasks. The number in each cell implies the number of problem instance solved by the approach of the row, but not solved by the approach of the column.

	BaB-Baseline	$\alpha\beta$ -Crown	NeuralSAT	Oliva ^{GR}	Oliva ^{SA}
BaB-Baseline	0	80	59	8	13
$\alpha\beta$ -Crown	111	0	11	23	39
NeuralSAT	126	47	0	30	46
Oliva ^{GR}	271	255	226	0	40
Oliva ^{SA}	247	242	213	11	0

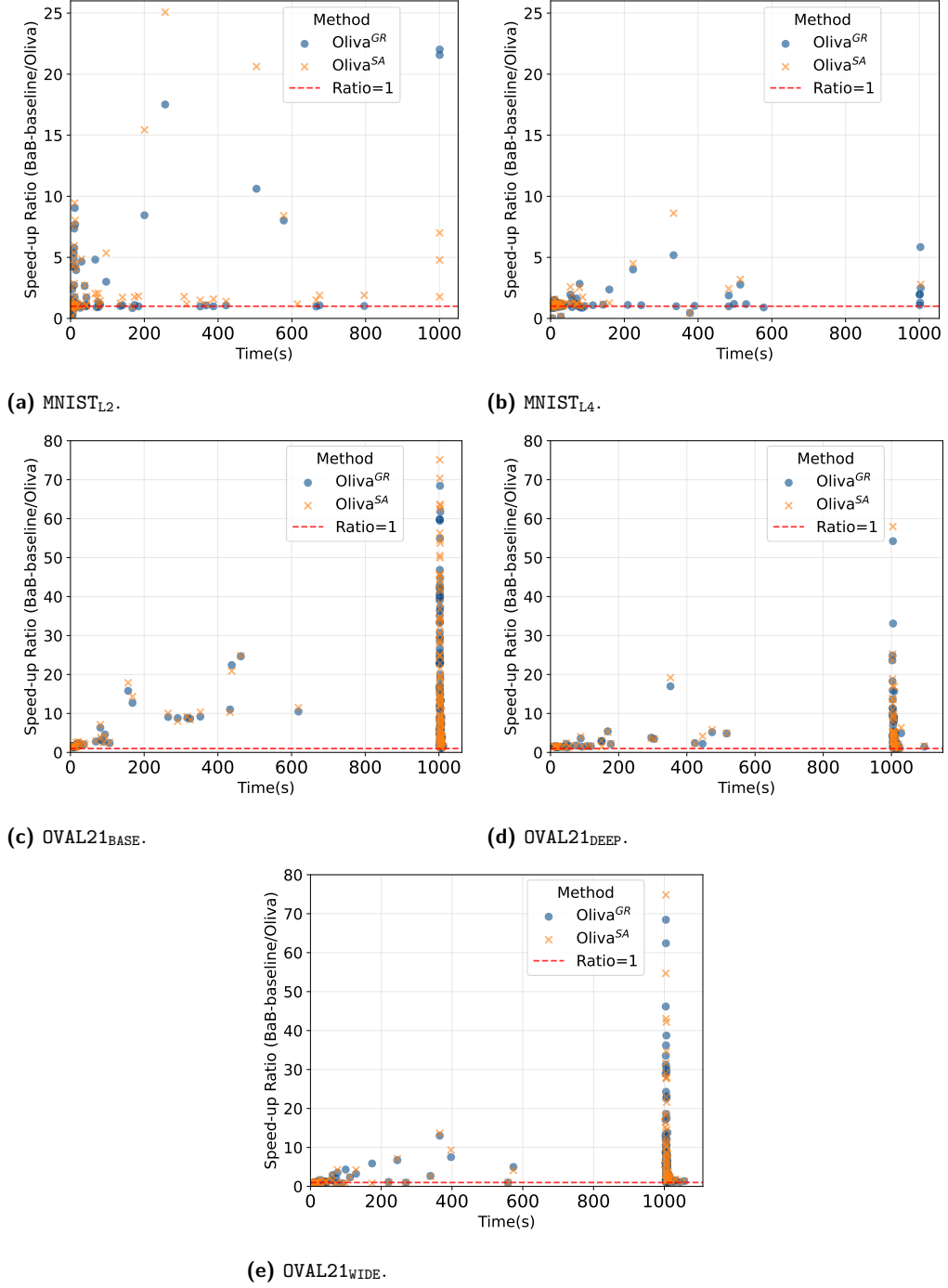
outperforms Oliva^{SA}, in that it solves 40 additional problems than Oliva^{SA}, although there are 11 problems Oliva^{SA} solves but Oliva^{GR} does not do. That said, it also demonstrates the complementary strengths between different approaches, namely, there is no one global optimal approach that can solve all the problems, and so it is worthwhile to try different approaches when one approach does not work.

In Fig. 5, we take a further comparison between Oliva and BaB-baseline, and draw a scatter plot to show the individual instances for which Oliva outperform BaB-baseline. The x -axis represents the time taken by the BaB-baseline method in seconds, while the y -axis shows the speedup ratio of Oliva over BaB-baseline. This ratio is calculated as $\tau_{\text{BaB}}(i)/\tau_{\text{Oliva}}(i)$, where $\tau_{\text{BaB}}(i)$ and $\tau_{\text{Oliva}}(i)$ denote the time taken by BaB-baseline and Oliva on the instance i , respectively. The blue dots and orange crosses represent all of the individual problem instances. The red line is the threshold at $1\times$ speedup, above which the ratio distinguishes that Oliva outperforms than BaB-baseline.

Across all the models, it can be seen that a significant number of blue dots (Oliva^{GR}) and orange crosses (Oliva^{SA}) appear above the red line, indicating that Oliva outperforms the BaB-baseline. Notably, around the 1000-second mark on the x -axis, there are multiple instances where the speedup ratio is up to $80\times$, meaning that Oliva can solve these problems more than 80 times faster than the BaB-baseline. This trend is particularly evident in OVAL21 models (OVAL21_{BASE}, OVAL21_{DEEP}, and OVAL21_{WIDE}) that have relatively complex architectures.

In Table 4, we summarize the statistical information of speedup ratios of our approaches over BaB-baseline. Overall, the two variants of Oliva achieve consistent performance improvements, with median speedups exceeding 2 times and mean gain values around 7 times.

Similar to simulated annealing, our tool Oliva^{SA} is a stochastic approach that is subject to randomness. To analyze the impact of randomness on the performance of Oliva^{SA}, we conduct five independent attempts with all the verification problems and compare them with the deterministic Oliva^{GR}. Since Oliva^{SA} incorporates a simulated annealing approach that makes probabilistic decisions during sub-problem exploration, different runs may explore the verification space in different orders, potentially leading to variations in performance.

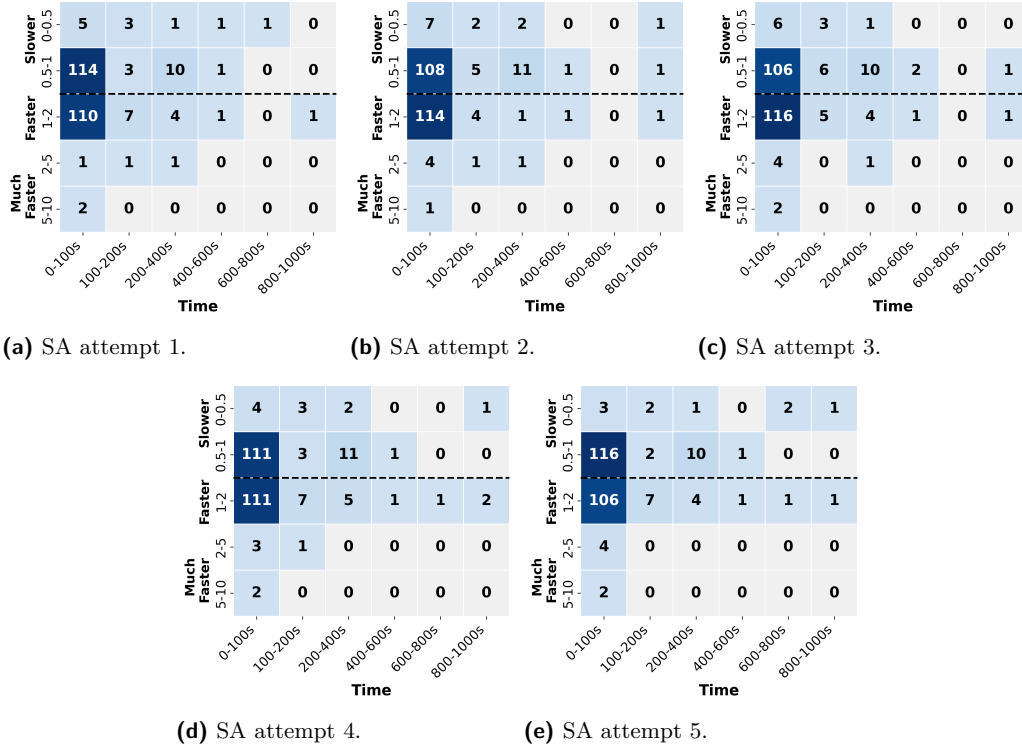


■ **Figure 5** RQ1 – Comparison of time cost and speedup ratio of the two variants $Oliva^{GR}$ and $Oliva^{SA}$ of our proposed approach Oliva over BaB-baseline.

The results are presented in Fig. 6, in which we only present the problems that are either solved by $Oliva^{GR}$ or $Oliva^{SA}$. Overall, the performance comparisons between $Oliva^{GR}$ and $Oliva^{SA}$ are consistent across different attempts, and are concentrated in the $0.5\text{--}2\times$ speedup bracket. This result suggests that the randomness in the simulated annealing approach does

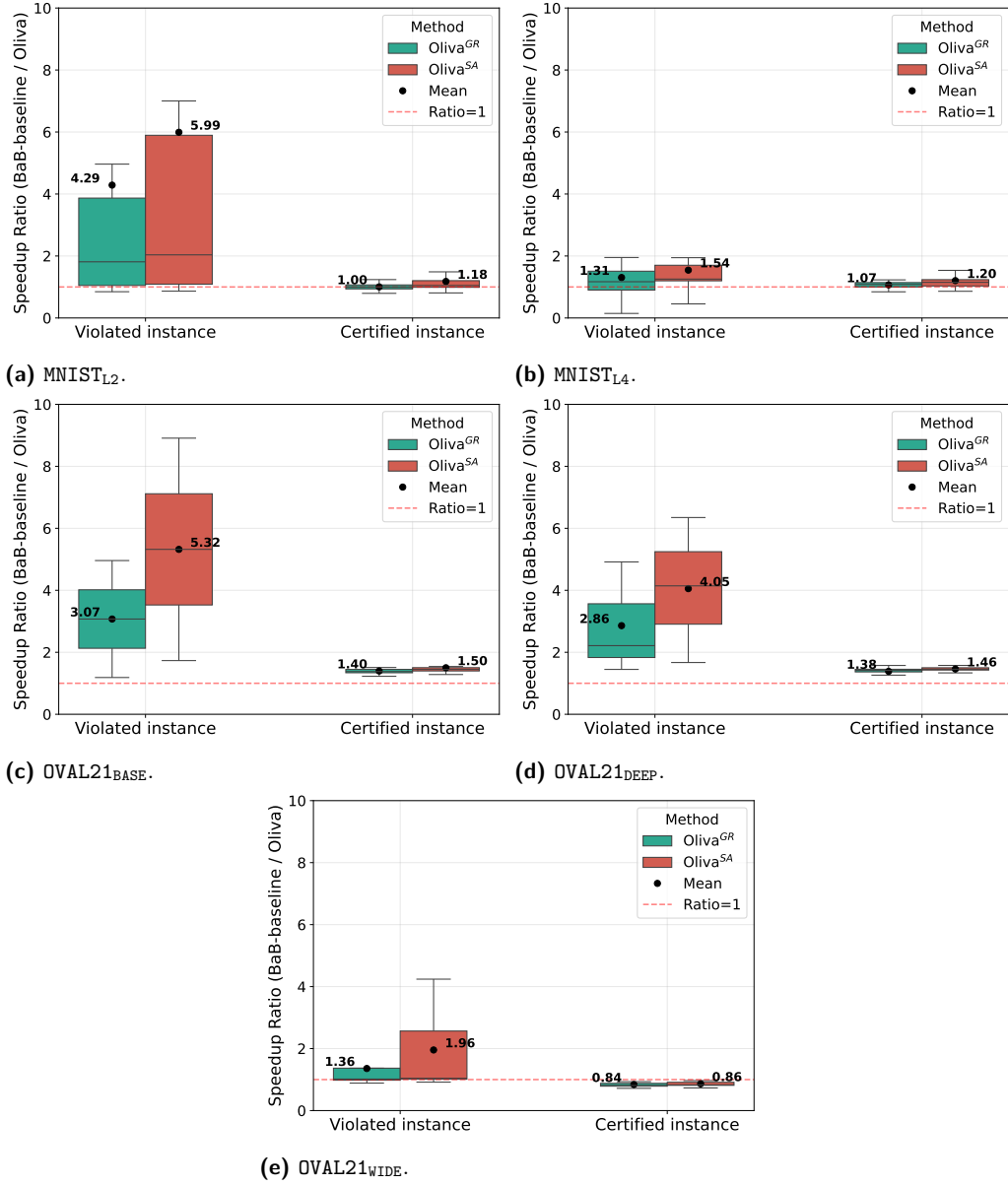
■ **Table 4** RQ1 – Statistics of speedup ratios of two variants of Oliva over BaB-baseline.

Model	Tool name	Min	Max	Median	Mean
Overall	Oliva ^{GR}	0.02	80.97	2.21	7.27
	Oliva ^{SA}	0.03	75.13	2.18	7.57
MNIST _{L2}	Oliva ^{GR}	0.11	22.02	1.04	2.49
	Oliva ^{SA}	0.11	25.07	1.12	2.50
MNIST _{L4}	Oliva ^{GR}	0.02	5.84	1.10	1.38
	Oliva ^{SA}	0.03	8.62	1.17	1.47
OVAL21 _{BASE}	Oliva ^{GR}	1.07	80.97	6.93	13.73
	Oliva ^{SA}	1.28	75.13	7.02	13.96
OVAL21 _{DEEP}	Oliva ^{GR}	1.01	54.23	2.80	5.37
	Oliva ^{SA}	1.03	57.96	2.34	5.31
OVAL21 _{WIDE}	Oliva ^{GR}	0.81	68.44	2.95	7.48
	Oliva ^{SA}	0.78	74.83	2.55	7.60



■ **Figure 6** The distributions of conclusive instances by the variant of Oliva^{SA}, in different speedups and time costs compared with Oliva^{GR}. The black dashed line indicates the performance of Oliva^{GR}: above the black dashed line, Oliva^{GR} outperforms Oliva^{SA}; below it, Oliva^{SA} performs better.

not substantially impact the effectiveness of Oliva^{SA}. Notably, in Fig 6c and Fig 6d, with the nature of the randomness, Oliva^{SA} exhibits the strength in finding solutions in the time cost of 100-600s that might be difficult to solve by the deterministic approach of Oliva^{GR}.

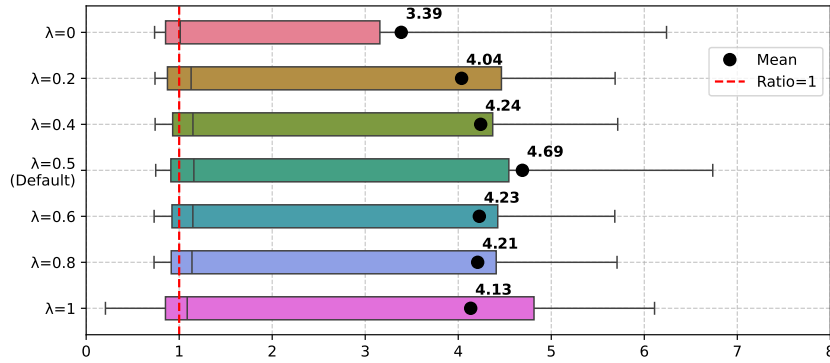


■ **Figure 7** RQ2 – Comparison between BaB-baseline and Oliva for violated and certified verification problem instances.

RQ2: How effective is Oliva in handling violated and certified problem instances, respectively?

Fig. 7 shows the performance advantage of Oliva^{GR} and Oliva^{SA} over BaB-baseline across different verification tasks. For violated instances (i.e., confirmed counterexample by BaB-baseline and Oliva), Oliva^{GR} and Oliva^{SA} consistently demonstrate superior efficiency, evidenced by lower median time costs and smaller interquartile ranges across all models.

In Fig. 7a, Oliva shows notably lower median time costs and compressed interquartile ranges compared to BaB-baseline, indicating more consistent and faster execution. In OVAL21_{BASE} and OVAL21_{DEEP} (Fig. 7c and Fig. 7d), Oliva^{GR} and Oliva^{SA} sub-problem selection



■ **Figure 8** RQ3 – Speedup over BaB-baseline under different hyperparameter λ values on verification tasks with OVAL21_{WIDE}.

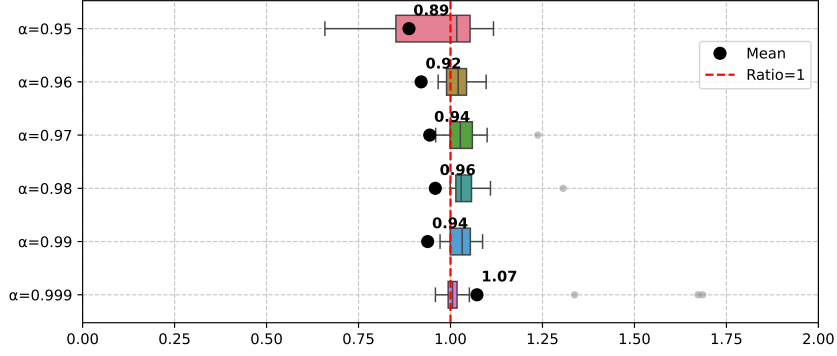
strategy is highly effective in problem instances where counterexamples exist. For OVAL21_{BASE}, Oliva^{GR} achieves a median speed-up ratio of approximately $2.5\times$ for violated instances, and Oliva^{SA} with reaching up to $4\times$ faster execution than BaB-baseline. Similarly, for OVAL21_{DEEP}, the median speed-up ratio is around $3\times$, with peak performance showing up to $5\times$ acceleration.

In contrast, for certified instances (i.e., confirmed robust), two variants of Oliva are generally on par with the BaB-baseline, indicating that in these cases, the performance of Oliva is comparable with BaB-baseline, which does not introduce overhead costs. This is expected, because in BaB, most of the time consumption is devoted to the problem solving process for each sub-problem. While our proposed approach introduces little overhead in the selection of the sub-problems to proceed with, the overhead is almost invisible.

RQ3: How does the hyperparameter λ and α in the order influences the performance of Oliva?

Fig. 8 highlights the impact of the λ parameter on the speedup of Oliva^{GR} over BaB-baseline for the verification task on OVAL21_{WIDE} model. In this experiment, we randomly select 20% of the instances, including 13 conclusive and 12 unknown cases, based on the BaB-baseline performance. The chosen instances are at varying levels of time consumption. This analysis examines λ values ranging from 0.0 to 1.0, and summarizes the speedup ratios of Oliva^{GR} under different λ values. The findings reveal that, first, the performances of Oliva^{GR} under different λ are relatively stable, mostly outperforming BaB-baseline. This implies that both of the attributes, including the level of problem splitting and the verifier assessment, are effective in guiding the space exploration. Moreover, the performance slightly improves as λ increases from 0.0 to 0.5, peaking at a speedup of 4.69. Beyond $\lambda = 0.5$, speedup declines, indicating 0.5 as the optimal value. In terms of the average improvement, the speedup vary from 3.39 to 4.69 and is not negligible, which emphasizes the importance of careful tuning of λ to maximize the efficiency of Oliva^{GR} in verification tasks.

Then, we settle $\lambda = 0.5$ as the default value to evaluate the temperature reduction parameter α , which ranges from 0.95 to 0.999, as suggested by [24]. Fig. 9 compares the speedup of Oliva^{SA} over Oliva^{GR}. The hyperparameter α determines the speed of “temperature” decreases during the verification processes, directly affecting the balances between exploration and exploitation. First, by the medians, we find that Oliva^{SA} under all α outperform Oliva^{GR}. Specifically, with $\alpha = 0.999$, Oliva^{SA} achieves better performance than Oliva^{GR} in average,



■ **Figure 9** RQ3 – Speedup over Oliva^{GR} under hyperparameter α values on verification tasks with $\text{OVAL21}_{\text{WIDE}}$.

which is a very slow cooling schedule (α close to 1) that allows the algorithm to thoroughly explore the search space early on before transitioning to exploitation. As α decreases, the temperature drops more rapidly, giving the algorithm less time to explore before converging on promising areas. The lower performance ratios seen with $\alpha = 0.95 - 0.98$ suggest that this faster cooling schedule may be too aggressive, causing Oliva^{SA} to commit too quickly to certain branches of the verification tree without adequately exploring alternatives. By our inspection of experimental results, the relative low mean values are caused by some specific cases, for which Oliva^{SA} does not perform well. Notably, when selecting $\alpha = 0.99$, while its mean ratio of 0.94 might seem underwhelming at a first glance, its median performance is notably strong compared to other α values, representing a balanced probability that allows sufficient exploration while maintaining steady progress toward exploitation. In practical verification tasks, as consistency can be as a valuable property, $\alpha = 0.99$ can thus be preferable despite its lower mean ratio.

6 Related Work

Neural Network Verification

Neural network verification has been extensively studied in the past few years, giving birth to many practical approaches [47, 23, 14, 22, 41, 44, 43, 29, 49, 39]. Approximated verifiers are preferable due to their efficiency, and many works aim to seek for tighter bounds for approximation refinement [2, 46, 40, 32, 31, 54, 27, 36, 15, 13, 9]. Some works aim to refine the approximation [55, 34, 20, 12, 60] by exploiting information from (spurious) counterexamples (known as CEGAR approach). In contrast, our approach does not aim to refine the approximation of the backend verifiers, but we estimate the probability of finding counterexamples in different sub-problems to efficiently explore the sub-problem space.

Problem Splitting Strategies in BaB

Problem splitting [38, 26, 16] is an important component in the BaB workflow, which is critical to the efficiency of verification [4]. Early research considers splitting input space by its dimensions [50, 1, 37]; however, that can lead to an exponential growth in the number of sub-problems w.r.t. the number of dimensions, which is intractable especially for applications that have high-dimensional data, such as image classification. More recently, many works start to perform problem splitting by decomposing ReLU functions, which can achieve

better performance [5] and scales better than input domain splitting. In these works, the ReLU selection strategy that decides the next neuron (i.e., the ReLU function) to be split is important and there emerge many effective strategies including DeepSplit [21] (which we adopted), BaBSR [5] and FSB [10]. Compared to those works, our approach is different and is orthogonal to them, because instead of selecting ReLU, we select the sub-problems to proceed with, which changes the naive breadth-first strategy of BaB in visiting the sub-problems. Notably, our work is in line with [57]; however, unlike [57] that targets incremental neural network verification, we target the problem of classic neural network verification, and we adopt a different stochastic optimization framework, i.e., simulated annealing.

Testing and Attacks

Testing is another effective quality assurance for neural networks, and it has also been extensively studied, such as [35, 33, 17, 45]. A similar line of works are adversarial attacks, that aim to generate adversarial examples [18, 3, 53, 58, 6] to fool the neural networks. While these approaches are efficient, they cannot provide rigorous guarantee on specification satisfaction, when they cannot find a counterexample, because they solve the problem by checking infinitely many single inputs in the input space. In comparison, our approach is sound, in the sense that if there does not exist a counterexample in the network, we can finally reach the certification of the network. This is because by our problem splitting, we deal with finitely many sub-problems.

7 Conclusion and Future Work

In this paper, we propose an approach Oliva with two variants Oliva^{GR} and Oliva^{SA} in order to achieve high efficiency of neural network verification. Oliva introduces a severity order over the sub-problems produced by BaB. This order, called counterexample potentiality, estimates the suspiciousness of each sub-problem in the sense how likely it contains a counterexample, based on both the level of problem splitting and the assessment from approximated verifiers that signify how far a sub-problem is from being violated. By prioritizing the sub-problems with higher counterexample potentiality, Oliva can efficiently reach verification conclusions – either finding a counterexample quickly or certifying the neural network without significant performance degradation. Specifically, our two variants of Oliva implement different strategies: *i*) Oliva^{GR} greedily exploits the most suspicious sub-problems; *ii*) Oliva^{SA}, inspired by simulated annealing, strikes a balance between exploitation of the promising sub-problems and exploration of the sub-problems that are less promising. Our experimental evaluation across 5 neural network models and 690 verification problems demonstrates that Oliva can achieve up to 80× speedup, compared to the state-of-the-art baseline approaches. Moreover, we also show the breakdown comparison for the certified problems and for the falsified problems, and we demonstrate that the performance advantages of the proposed approach indeed come from the strategy guided by counterexample potentiality.

To the best of our knowledge, our work is the first that exploits the power of stochastic optimization in neural network verification, by taking the sub-problems as the search space. In future, we plan to do more exploration in this direction, e.g., trying other stochastic optimization algorithms, such as genetic algorithms, and comparing the performances of different strategies, in order to deliver better verification approaches.

References

- 1 Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, pages 731–744, 2019. doi:10.1145/3314221.3314614.
- 2 R. Anderson, J Huchette, C Tjandraatmadja, and JP Vielma. Strong convex relaxations and mixed-integer programming formulations for trained neural networks (2018), 1811.
- 3 Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *European conference on computer vision*, pages 484–501. Springer, 2020. doi:10.1007/978-3-030-58592-1_29.
- 4 Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Branch and bound for piecewise linear neural network verification. *CoRR*, abs/1909.06588, 2019. doi:10.48550/arXiv.1909.06588.
- 5 Rudy Bunel, P Mudigonda, Ilker Turkaslan, Philip Torr, Jingyue Lu, and Pushmeet Kohli. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020. URL: <https://jmlr.org/papers/v21/19-468.html>.
- 6 Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017. doi:10.1109/SP.2017.49.
- 7 Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In Deepak D’Souza and K. Narayan Kumar, editors, *Automated Technology for Verification and Analysis*, pages 251–268. Springer Int. Publishing, 2017. doi:10.1007/978-3-319-68167-2_18.
- 8 Moumita Das, Rajarshi Ray, Swarup Kumar Mohalik, and Ansuman Banerjee. Fast falsification of neural networks using property directed testing. *arXiv preprint arXiv:2104.12418*, 2021. doi:10.48550/arXiv.2104.12418.
- 9 Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy R Bunel, Shreya Shankar, Jacob Steinhardt, Ian Goodfellow, Percy S Liang, et al. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Advances in Neural Information Processing Systems*, 33:5318–5331, 2020.
- 10 Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip HS Torr, and M Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition. *arXiv preprint arXiv:2104.06718*, 2021.
- 11 Hai Duong, Linhan Li, ThanhVu Nguyen, and Matthew B. Dwyer. A DPLL(T) framework for verifying deep neural networks. *CoRR*, abs/2307.10266, 2023. doi:10.48550/arXiv.2307.10266.
- 12 Hai Duong, Dong Xu, ThanhVu Nguyen, and Matthew B Dwyer. Harnessing neuron stability to improve dnn verification. *Proceedings of the ACM on Software Engineering*, 1(FSE):859–881, 2024. doi:10.1145/3643765.
- 13 Krishnamurthy Dj Dvijotham, Robert Stanforth, Sven Gowal, Chongli Qin, Soham De, and Pushmeet Kohli. Efficient neural network verification with exactness characterization. In *Uncertainty in artificial intelligence*, pages 497–507. PMLR, 2020.
- 14 Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis: 15th Int. Symp., ATVA 2017, Proceedings 15*, pages 269–286. Springer, October 2017.
- 15 Mahyar Fazlyab, Manfred Morari, and George J Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Transactions on Automatic Control*, 67(1):1–15, 2020.
- 16 Claudio Ferrari, Mark Niklas Muller, Nikola Jovanovic, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. *arXiv preprint arXiv:2205.00263*, 2022. doi:10.48550/arXiv.2205.00263.

- 17 Xiang Gao, Ripon K Saha, Mukul R Prasad, and Abhik Roychoudhury. Fuzz testing based data augmentation to improve robustness of deep neural networks. In *Proceedings of the acm/ieee 42nd international conference on software engineering*, pages 1147–1158, 2020. doi:10.1145/3377811.3380415.
- 18 Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *3rd Int. Conf. on Learning Representations (ICLR'15)*, San Diego, CA, United States, 2015. Int. Conf. on Learning Representations, ICLR.
- 19 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: <https://www.gurobi.com>.
- 20 Karthik Hanumanthaiah and Samik Basu. Iterative counter-example guided robustness verification for neural networks. In *International Symposium on AI Verification*, pages 179–187. Springer, 2024. doi:10.1007/978-3-031-65112-0_9.
- 21 Patrick Henriksen and Alessio Lomuscio. Deepsplit: An efficient splitting method for neural network verification via indirect effect analysis. In *IJCAI*, pages 2549–2555, 2021. doi:10.24963/ijcai.2021/351.
- 22 Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Computer Aided Verification: 29th Int. Conf., CAV 2017, Part I 30*, pages 3–29. Springer, July 2017. doi:10.1007/978-3-319-63387-9_1.
- 23 Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčák, editors, *Computer Aided Verification*, pages 97–117. Springer Int. Publishing, 2017. doi:10.1007/978-3-319-63387-9_5.
- 24 Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983. doi:10.1126/science.220.4598.671.
- 25 Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021. doi:10.1561/24000000035.
- 26 Jingyue Lu and M Pawan Kumar. Neural network branching for neural network verification. *arXiv preprint arXiv:1912.01329*, 2019. arXiv:1912.01329.
- 27 Zhongkui Ma, Jiaying Li, and Guangdong Bai. Relu hull approximation. *Proceedings of the ACM on Programming Languages*, 8(POPL):2260–2287, 2024. doi:10.1145/3632917.
- 28 Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th Int. Conf. on Learning Representations (ICLR'18)*, Vancouver, Canada, 2018.
- 29 Christoph Müller, Gagandeep Singh, Markus Püschel, and Martin T Vechev. Neural network robustness verification on gpus. *CoRR, abs/2007.10868*, 2020.
- 30 Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T Johnson. 3rd international verification of neural networks competition (VNN-COMP 2022): Summary and results. *arXiv preprint arXiv:2212.10376*, 2022. doi:10.48550/arXiv.2212.10376.
- 31 Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. Precise multi-neuron abstractions for neural network certification. *arXiv preprint arXiv:2103.03638*, 2021. arXiv:2103.03638.
- 32 Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. Prima: general and precise neural network certification via scalable convex hull approximations. *ACM on Programming Languages*, 6(POPL):1–33, 2022. doi:10.1145/3498704.
- 33 Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*, pages 4901–4911. PMLR, 2019. URL: <http://proceedings.mlr.press/v97/odena19a.html>.

- 34 Matan Ostrovsky, Clark Barrett, and Guy Katz. An abstraction-refinement approach to verifying convolutional neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 391–396. Springer, 2022. doi:10.1007/978-3-031-19992-9_25.
- 35 Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *26th Symp. on Operating Systems Principles*, pages 1–18, 2017. doi:10.1145/3132747.3132785.
- 36 Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. *Advances in neural information processing systems*, 31, 2018.
- 37 Vicenc Rubies-Royo, Roberto Calandra, Dusan M Stipanovic, and Claire Tomlin. Fast neural network verification via shadow prices. *arXiv preprint arXiv:1902.07247*, 2019. arXiv:1902.07247.
- 38 Zhouxing Shi, Qirui Jin, Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Neural network verification with branch-and-bound for general nonlinearities. *arXiv preprint arXiv:2405.21063*, 2024. doi:10.48550/arXiv.2405.21063.
- 39 Zhouxing Shi, Yihan Wang, Huan Zhang, J Zico Kolter, and Cho-Jui Hsieh. Efficiently computing local lipschitz constants of neural networks via bound propagation. *Advances in Neural Information Processing Systems*, 35:2350–2364, 2022.
- 40 Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. *Advances in Neural Information Processing Systems*, 32, 2019.
- 41 Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, volume 31, 2018.
- 42 Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. *Advances in neural information processing systems*, 31, 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/f2f446980d8e971ef3da97af089481c3-Abstract.html>.
- 43 Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. In *Int. Conf. on learning representations*, 2018.
- 44 Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *ACM on Programming Languages*, 3(POPL):1–30, 2019. doi:10.1145/3290354.
- 45 Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *40th Int. Conf. on Software Engineering*, pages 303–314, 2018. doi:10.1145/3180155.3180220.
- 46 Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Kishor Patel, and Juan Pablo Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *Advances in Neural Information Processing Systems*, 33:21675–21686, 2020.
- 47 Vincent Tjeng, Kai Y Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *Int. Conf. on Learning Representations*, 2018.
- 48 Shubham Ugare, Debangshu Banerjee, Sasa Misailovic, and Gagandeep Singh. Incremental verification of neural networks. *ACM on Programming Languages*, 7(PLDI):1920–1945, 2023. doi:10.1145/3591299.
- 49 Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *Advances in neural information processing systems*, 31, 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/2ecd2bd94734e5dd392d8678bc64cdab-Abstract.html>.

- 50 Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symp. (USENIX Security 18)*, pages 1599–1614, 2018. URL: <https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi>.
- 51 Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems*, 34:29909–29921, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/fac7fead96dafceaf80c1daffeae82a4-Abstract.html>.
- 52 Haoze Wu, Omri Isac, Aleksandar Zeljic, Teruhiro Tagomori, Matthew L. Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, Pei Huang, Ori Lahav, Min Wu, Min Zhang, Ekaterina Komendantskaya, Guy Katz, and Clark W. Barrett. Marabou 2.0: A versatile formal analyzer of neural networks. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24–27, 2024, Proceedings, Part II*, volume 14682 of *Lecture Notes in Computer Science*, pages 249–264. Springer, Springer, 2024. doi:10.1007/978-3-031-65630-9_13.
- 53 Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, and Alan L Yuille. Improving transferability of adversarial examples with input diversity. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2730–2739, 2019. doi:10.1109/CVPR.2019.00284.
- 54 Xiaoyong Xue, Xiyue Zhang, and Meng Sun. kprop: Multi-neuron relaxation method for neural network robustness verification. In *International Conference on Fundamentals of Software Engineering*, pages 142–156. Springer, 2023. doi:10.1007/978-3-031-42441-0_11.
- 55 Pengfei Yang, Renjue Li, Jianlin Li, Cheng-Chao Huang, Jingyi Wang, Jun Sun, Bai Xue, and Lijun Zhang. Improving neural network verification through spurious region guided refinement. In *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pages 389–408. Springer, 2021. doi:10.1007/978-3-030-72016-2_21.
- 56 Yang Yu, Hong Qian, and Yi-Qi Hu. Derivative-free optimization via classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. doi:10.1609/aaai.v30i1.10289.
- 57 Guanqin Zhang, Zhenya Zhang, H.M.N. Dilum Bandara, Shiping Chen, Jianjun Zhao, and Yulei Sui. Efficient incremental verification of neural networks guided by counterexample potentiality. *Proc. ACM Program. Lang.*, 9(OOPSLA1), April 2025. doi:10.1145/3720417.
- 58 Huan Zhang, Shiqi Wang, Kaidi Xu, Yihan Wang, Suman Jana, Cho-Jui Hsieh, and Zico Kolter. A branch and bound framework for stronger adversarial attacks of relu networks. In *International Conference on Machine Learning*, pages 26591–26604. PMLR, 2022. URL: <https://proceedings.mlr.press/v162/zhang22ae.html>.
- 59 Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems*, 31, 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/d04863f100d59b3eb688a11f95b0ae60-Abstract.html>.
- 60 Zhe Zhao, Yedi Zhang, Guangke Chen, Fu Song, Taolue Chen, and Jiaxiang Liu. Cleverest: accelerating cegar-based neural network verification via adversarial attacks. In *International Static Analysis Symposium*, pages 449–473. Springer, 2022. doi:10.1007/978-3-031-22308-2_20.