Faster Verified Explanations for Neural Networks

Alessandro De Palma[1]
London School of Economics and Political Science, UK

Greta Dolcetti
Ca' Foscari University of Venice, Italy

Caterina Urban
Inria & ENS | PSL, France

Abstract

Verified explanations are a theoretically-principled way to explain the decisions taken by neural networks, which are otherwise black-box in nature. However, these techniques face significant scalability challenges, as they require multiple calls to neural network verifiers, each of them with an exponential worst-case complexity. We present FaVeX, a novel algorithm to compute verified explanations. FaVeX accelerates the computation by dynamically combining batch and sequential processing of input features, and by reusing information from previous queries, both when proving invariances with respect to certain input features, and when searching for feature assignments altering the prediction. Furthermore, we present a novel and hierarchical definition of verified explanations, termed verifier-optimal robust explanations, that explicitly factors the incompleteness of network verifiers within the explanation. Our comprehensive experimental evaluation demonstrates the superior scalability of both FaVeX, and of verifier-optimal robust explanations, which together can produce meaningful formal explanation on networks with hundreds of thousands of non-linear activations.

2012 ACM Subject Classification Theory of computation → Program analysis; Theory of computation → Abstraction; Mathematics of computing → Continuous optimization

Keywords and phrases Explainable Machine Learning, Neural Network Verification

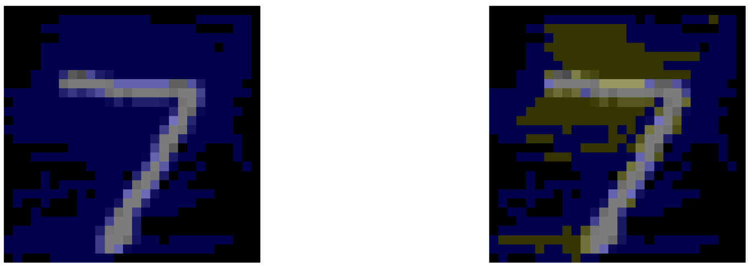
1 Introduction

Machine learning models have demonstrated remarkable performance across a wide range of tasks, from image classification to natural language processing. Yet, as these models are increasingly deployed in safety-critical domains – such as finance, transportation, and even healthcare – concerns about trust, accountability, and robustness become paramount. In such contexts, explainability is essential: users and auditors must understand *why* a model produces a particular prediction, and whether that prediction remains stable under small, semantically meaningful perturbations of the input [19, 14].

A promising line of research in eXplainable Artificial Intelligence addresses this need through *verified explanations* [42, 43, etc.], which aim to formally certify which input features are responsible for a model prediction by leveraging the theory and tools of machine learning verification [1, 60]. Thus, verified explanations aim to provide rigorous guarantees, in contrast to heuristic or gradient-based methods [22, 39, 49, etc.]. Among verified explanations, *(optimal) robust explanations* [27, 34, 65, 25] have emerged as principled and mathematically grounded notions linking local robustness to the minimality of feature sets preserving a model prediction: they identify subsets of input features that, if left unchanged, cannot alter the predicted outcome. Formally, this corresponds to establishing local robustness of the model with respect to perturbations on the remaining input features.

However, despite their theoretical appeal, optimal robust explanations are difficult to scale beyond small neural networks with tens of non-linear activations or low-dimensional

[1] Work partly carried out at Inria & ENS | PSL.



(a) Robust explanation. **(b)** Verifier-optimal robust explanation.

Figure 1 Comparison of standard (a) and verifier-optimal (b) robust explanations ($\epsilon = 0.25$) of the first image in the MNIST testing dataset, where *counterfactuals* are highlighted in *yellow*, *unknowns* in *blue*, and *invariants* are not highlighted.

input spaces [65, 64]. Indeed, the computation of optimal robust explanations requires repeatedly querying neural network verifiers, which solve NP-hard problems [32] and often exhibit exponential worst-case complexity. In practice, even finding a single counterfactual – a necessary step to ensure the *optimality* of an explanation – can quickly become infeasible for state-of-the-art neural networks with hundreds of thousands of non-linear activations (we demonstrate and discuss this in detail in Section 7.2).

Moreover, crucially, the definition of optimal robust explanations assumes that verification is complete, i.e., always either proving robustness or producing a counterfactual. This assumption is unrealistic in practice because, due to their exponential worst-case complexity, modern verifiers have to routinely enforce timeouts to handle large models efficiently, de facto making compromises about completeness to gain better performance [5, 32, 58]. As a consequence, although optimal explanations are a useful semantic ideal, they are rarely practically achievable for modern architectures. For instance, Figure 1a shows the robust (but not optimal) explanation that can be computed in practice for the classification of the first image in the MNIST testing data set by a state-of-the-art convolutional neural network (with roughly $230k$ ReLUs) from the certified training literature [12]. Notably, the explanation (i.e., the pixels that are highlighted in blue) covers a large portion of the image, making it too coarse to provide meaningful insights in practical settings.

Contributions. We address the aforementioned challenges through three main contributions:

1. **Verifier-Optimal Robust Explanations.** We introduce a new, practically achievable counterpart of optimal robust explanations – verifier-optimal robust explanations – that explicitly incorporate the behavior of a given verifier and its underlying practical compromises on completeness (Section 4). Our notion partitions the input features into (i) *invariants*, for which the verifier proves robustness, (ii) *counterfactuals*, for which the verifier identifies perturbations causing misclassification, and (iii) *unknowns*, where verification remains inconclusive. This viewpoint better reflects what can be guaranteed in practice and enables the discovery of counterfactuals at a scale unattainable with prior definitions. Figure 1b shows the verifier-optimal robust explanation computed by our approach for the same MNIST image and convolutional model of Figure 1a. In this case, the explanation is *hierarchical*, with the most informative pixels (i.e., the counterfactuals, highlighted in yellow) covering only a small portion of the image. Importantly, the definition of verifier-optimal robust explanation is a generalization of that of optimal robust explanations: when the verifier is complete, the set of unknowns is empty and verifier-optimal robust explanations coincide exactly with optimal robust explanations (Lemma 4).

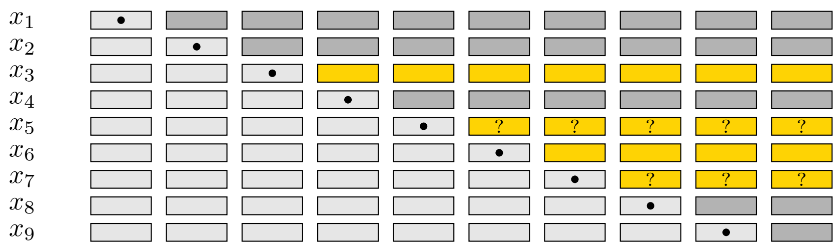


Figure 2 Example of computing a standard (optimal) robust explanation on 9 features. The verification steps proceed from left to right; the dotted box indicates the feature under analysis, dark grey shows the invariants, yellow shows the explanations for which a counterexample has been found, and question marks denote features for which a timeout has been encountered. The final results of the analysis can be seen in the rightmost column.

2. **FaVeX.** We propose a new algorithmic methodology that substantially accelerates the computation of both standard and verifier-optimal explanations. FaVeX (Section 5) combines (a) batch-based processing of robustness queries with a binary search-based mechanism, (b) a principled fallback to sequential feature evaluation, and (c) two orthogonal speedup strategies: branch reuse within branch-and-bound verification (Section 5.2), and restricted-space adversarial attacks to accelerate counterfactual search (Section 5.3). We also introduce a traversal strategy aligned with the verifier's logit-difference objective, improving explanation compactness (Section 5.1).
3. **Implementation and Experimental Evaluation.** We implemented our algorithm using the PyTorch deep learning library, employing the OVAL branch-and-bound framework [6, 4, 11, 10] as the backbone for our verifier (Section 6). We run our experimental evaluation (Section 7) on state-of-the-art neural networks trained on popular datasets with a varying number of parameters and different architectures. The results demonstrate that FaVeX significantly reduces the time required to compute standard robust explanations on small fully-connected networks, while enabling verifier-optimal robust explanations to be computed efficiently even on larger convolutional networks. Importantly, FaVeX can find counterfactuals even for networks with hundreds of thousands of non-linear activations, making formal explanations practical and meaningful at scale.

2 Optimal vs. Verifier-Optimal Robust Explanations

In this section, through a simple example, we provide an informal overview of the difference between our verifier-optimal robust explanations and (optimal) robust explanations.

Let us consider a classifier with 9-dimensional input and let $\mathbf{x} = (x_1, \dots, x_9)$ be the input vector for which we want to compute an explanation. For simplicity, we employ a basic deletion-based algorithm [65, 25] that processes input features indexes in sequential order from 1 to 9. Each step introduces ϵ-bounded ℓ_∞ perturbations to the input feature under analysis and selected previous features (depending on the explanation definition), while all remaining input features remain unchanged. A verifier is then queried to determine whether this perturbation is robust (i.e., the model's prediction remains the same), a counterexample is found (i.e., applying the perturbation causes the model to change its prediction), or a timeout is reached (i.e., the query cannot be resolved within the time limit).

Optimal Robust Explanations. Figure 2 shows how the analysis proceeds to compute a standard robust (but, in practice, not optimal) explanation. First, only x_1 is perturbed and

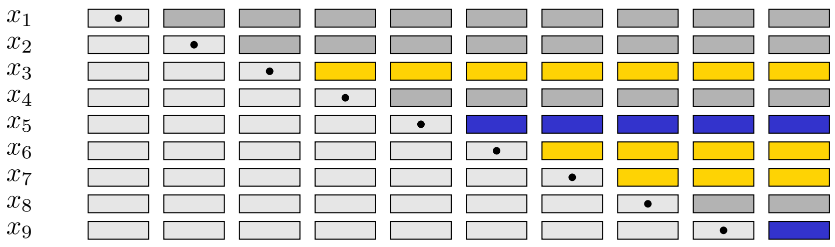


Figure 3 Example of computing a verifier-optimal robust explanation on 9 features. The verification steps proceed from left to right; the dotted box indicates the feature under analysis, dark grey shows the invariants, blue shows the unknowns, yellow shows the explanations for which a counterexample has been found, and question marks denote features for which a timeout has been encountered. The final results of the analysis can be seen in the rightmost column.

the verifier is queried. Suppose that robustness is verified; the feature index is then added to the invariants set $\mathcal{R}_{\mathbf{x}}$ (initially empty). Next, the perturbation is applied to the features indexed by $\mathcal{R}_{\mathbf{x}}$ (currently just x_1) and to the new feature under analysis, x_2. Suppose robustness is again verified; this feature index is also added to $\mathcal{R}_{\mathbf{x}}$. The analysis then proceeds with x_3. Suppose now that the verifier finds a counterexample. This feature is therefore considered part of the explanation and will not be perturbed in future steps. The analysis continues with x_4 which, after querying the verifier, we assume is considered an invariant. When x_5 is perturbed (along with the invariants x_1, x_2, and x_4), suppose that the verifier times out. Since the definition of optimal robust explanation does not account for the incompleteness caused by timeouts, the feature is considered part of the explanation from this point on. The computation continues until all input features have been processed. Suppose that a counterexample is found for x_6 and another timeout is encountered for x_7, while x_8 and x_9 are invariants. Features x_3, x_5, x_6, and x_7 are indiscriminately considered explanations. However, given the verifier timeouts, x_5 and x_7 may actually be invariants.

Verifier-Optimal Robust Explanations. Figure 3 instead illustrates the computation of a verifier-optimal robust explanation. In this case the analysis explicitly accounts for verifier limitations due to timeouts. Thus, rather than considering x_5 part of the explanation, the analysis adds the feature index to the unknowns set $\mathcal{U}_{\mathbf{x}}$ (initially empty). The analysis then proceeds by allowing perturbation also to the features indexed by $\mathcal{U}_{\mathbf{x}}$. This way, suppose that a counterexample is found for x_7 and a timeout is encountered for x_9 (which was instead considered invariant in Figure 2). This can occur because the considered perturbation region is effectively larger, as it also includes the features indexed by $\mathcal{U}_{\mathbf{x}}$. (Although this example is illustrative, such effects can also occur in practice. For instance, upon closer inspection, Figure 1b contains few more pixels highlighted in blue than Figure 1a.) The final explanation, containing the features x_3, x_5, x_6, x_7, x_9, is larger than the one in Figure 2 but it is *hierarchical*: features x_3, x_6, x_7 (counterfactuals) are more important to the classification outcome, while x_5 and x_9 (unknowns) are less important.

3 Background

In the following, we will denote vectors by boldface lowercase letters ($\mathbf{x} \in \mathbb{R}^d$), and use subscripts to index their entries (for instance, $\mathbf{x}_i$ denotes the i-th entry). Furthermore, we will denote sets by calligraphic uppercase letters (e.g, $\mathcal{A}$), and use $\mathbf{x}_{\mathcal{A}}$ to denote the vector containing the entries of $\mathbf{x}$ indexed by the elements of $\mathcal{A}$.

3.1 Neural Network Verification

Neural Networks. A *neural network* classifier is a function $f : \mathbb{R}^d \to \mathbb{R}^k$ mapping a d-dimensional input vector of *features* $\mathbf{x}$ to a k-dimensional output vector $f(\mathbf{x})$ of *logits*, i.e., unnormalized confidence scores. The classification output of a neural network classifier f is $y_f : \mathbb{R}^k \to \{1, \dots, k\}$ defined as $y_f(\mathbf{x}) \overset{\text{def}}{=} \operatorname{argmax}_i f(\mathbf{x})_i$.

Among the most widely used classifiers in modern deep learning are ReLU-activated neural networks, which consist of a sequential composition $f = f_l \circ \cdots \circ f_1$ of l layers, where each layer $f_i \colon \mathbb{R}^m \to \mathbb{R}^n$, is either an affine transformation $f_i(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, with $\mathbf{A} \in \mathbb{R}^{n \times m}$, $\mathbf{b} \in \mathbb{R}^n$, or an entry-wise applications of the rectified linear unit (ReLU) activation function, $\mathrm{ReLU}(x) \overset{\text{def}}{=} \max(0, x)$; the final layer f_l is an affine transformation.

Local Robustness. Neural network verification aims to provide formal guarantees about the neural network behavior. In particular, verifying neural network *safety*, involves proving that all network outputs corresponding to inputs satisfying a given input property also satisfy a specified output property. A widely studies safety property is *local robustness*, which ensure that small perturbations of an input do not change the classification output of a neural network. Formally, a neural network classifier $f : \mathbb{R}^d \to \mathbb{R}^k$ is said to be locally robust on an input $\mathbf{x} \in \mathbb{R}^d$ if, given a set of allowed perturbations $C(\mathbf{x}) \subseteq \mathbb{R}^d$, for all $\mathbf{x}' \in \mathbb{R}^d$ we have:

$$\mathbf{x}' \in C(\mathbf{x}) \Rightarrow y_f(\mathbf{x}') = y_f(\mathbf{x})$$

that is, the classification output of the neural network is invariant to perturbations in $C(\mathbf{x})$. In the following, we focus on perturbations within the ℓ_∞-ball with radius $\epsilon > 0$ centered at $\mathbf{x}$: $C(\mathbf{x}) = B^\epsilon(\mathbf{x}) \overset{\text{def}}{=} \{\mathbf{x}' \in \mathbb{R}^d \mid \|\mathbf{x}' - \mathbf{x}\|_\infty \leq \epsilon\}$.

In some cases, we are interested in perturbations that affect only a subset of the input features. Let $\mathcal{A} \subseteq \{1, \dots, d\}$ denote the indices of input features that may be perturbed. We will write $\overline{\mathcal{A}} := \{1, \dots, d\} \setminus \mathcal{A}$ for the complement of $\mathcal{A}$ under $\{1, \dots, d\}$. We define the set of allowed perturbations restricted to $\mathcal{A}$ as $B^\epsilon_{\mathcal{A}}(\mathbf{x}) \overset{\text{def}}{=} \{\mathbf{x}' \in \mathbb{R}^d \mid \mathbf{x}'_{\overline{\mathcal{A}}} = \mathbf{x}_{\overline{\mathcal{A}}} \land \|\mathbf{x}'_{\mathcal{A}} - \mathbf{x}_{\mathcal{A}}\|_\infty \leq \epsilon\}$.

We define a *robustness query* as a tuple $(\mathbf{x}, \mathcal{A}, \epsilon, f)$, where $\mathbf{x}$ denotes the input vector, $\mathcal{A}$ the set of the indices of the perturbed input features, ϵ the perturbation radius, and f the employed network. Specifically, $(\mathbf{x}, \mathcal{A}, \epsilon, f)$ amounts to assessing local robustness of f on $\mathbf{x}$ given the set of allowed perturbations $B^\epsilon_{\mathcal{A}}(\mathbf{x})$. Let $\mathcal{Q}$ be the set of all robustness queries. A *neural network verifier* $v : \mathcal{Q} \to \{-1, 0, 1\}$ is a function taking a robustness query as input, and returning: 1 if local robustness is proved; -1 if a *counterfactual* is found, that is, an input $\mathbf{x}' \in B^\epsilon_{\mathcal{A}}(\mathbf{x})$ such that $y_f(\mathbf{x}') \neq y_f(\mathbf{x})$; and 0 in case the verification is inconclusive. A *complete* verifier $\overline{v} : \mathcal{Q} \to \{-1, 1\}$ never returns 0, it always either verifies the query if it holds (returns 1), or finds a counterfactual (returns -1).

Internally, a neural network verifier v leverages an *analyzer* $a : \mathcal{Q} \to \mathbb{R}$ which, given a robustness query $(\mathbf{x}, \mathcal{A}, \epsilon, f)$, computes lower and upper bound values of each logit in $f(\mathbf{x})$ for any input vector $\mathbf{x}' \in B^\epsilon_{\mathcal{A}}(\mathbf{x})$. To do so it can leverage different abstractions such as IBP [17, 18], CROWN / DeepPoly [68, 56], or α-CROWN [66], among others [37, 46, 55, 54, 61, etc.]. Based on these bounds, a returns the lower bound value of the *worst-case logit difference*, the minimum difference between the logit corresponding to the true class $f(\mathbf{x})_{y_f(\mathbf{x})}$ and the logit corresponding to any other class: $\min_{i \neq y_f(\mathbf{x})}(f(\mathbf{x})_{y_f(\mathbf{x})} - f(\mathbf{x})_i)$. This lower bound is central to verification: if strictly positive, the logit corresponding to the true class remains larger than all others for every perturbed input vector in $B^\epsilon_{\mathcal{A}}(\mathbf{x})$, thus local robustness of f on x is verified (the verifier v can return 1).

Branch-and-Bound. With the state of the art in neural network verification, *branch-and-bound* [5, 11] has emerged as the dominant paradigm for designing complete verifiers [62, 69],

Algorithm 1 Branch-and-Bound

```
1: function $\textsc{BaB}(a, (\mathbf{x}, \mathcal{A}, \epsilon, f))$
                                                        ▷ $a\colon \hat{\mathcal{Q}} \to \mathbb{R}, (\mathbf{x}, \mathcal{A}, \epsilon, f) \in \mathcal{Q}$
2:     unresolved ← $\{(\mathbf{x}, \mathcal{A}, \epsilon, f, \emptyset)\}$          ▷ $(\mathbf{x}, \mathcal{A}, \epsilon, f, \emptyset) \in \hat{\mathcal{Q}}$
3:     for $Q \in$ unresolved do
4:         if $\textsc{cex}(\mathbf{x}, \mathcal{A}, \epsilon, f)$ then return $-1$
5:         unresolved ← unresolved $\setminus \{Q\}$
6:         result ← $a(Q)$                                  ▷ Bounding step
7:         if result $\leq 0$ then                            ▷ Branching step
8:             $Q_1, Q_2 \leftarrow \textsc{split}(Q)$
9:             unresolved ← unresolved $\cup \{Q_1, Q_2\}$
10:    return 1
```

by recursively partitioning the verification problem into more tractable subproblems.

Let $\hat{\mathcal{Q}} \overset{\text{def}}{=} \{(\mathbf{x}, \mathcal{A}, \epsilon, f, C) \mid (\mathbf{x}, \mathcal{A}, \epsilon, f) \in \mathcal{Q}, C \in \mathcal{P}(\mathcal{C})\}$ be an extension of $\mathcal{Q}$ where each robustness query $(\mathbf{x}, \mathcal{A}, \epsilon, f) \in \mathcal{Q}$ is augmented by a set of constraints $C \in \mathcal{P}(\mathcal{C})$. We refer to elements of $\hat{\mathcal{Q}}$ as *robustness subproblems*. Note that, an analyzer $a\colon Q \to \mathbb{R}$ naturally accommodates the additional constraints introduced by subproblems. Therefore, from now on, we assume that analyzers directly operate on subproblems: $a\colon \hat{Q} \to \mathbb{R}$. For a subproblem $Q = (\mathbf{x}, \mathcal{A}, \epsilon, f, C) \in \hat{\mathcal{Q}}$, we write Q_C to denote its associated constraint set C. Branch-and-bound partitions a given robustness subproblem $Q \in \hat{\mathcal{Q}}$ by growing its set of constraints Q_C. In the following, we consider the *ReLU splitting* [5, 11, 24] partitioning strategy. That is, we assume that constraints in $\mathcal{C}$ determine the sign of the pre-activation value of a ReLU: let $x_{i,j} = \max(0, \hat{x}_{i,j})$ denote a ReLU at the i-th layer and j-index of a neural network classifier; the constraint $\hat{x}_{i,j} \geq 0$ determines that the ReLU behaves as the identify function, while the constraint $\hat{x}_{i,j} < 0$ determines that the ReLU behaves as the constant function equal to zero. Another partitioning strategy in the literature is input splitting [2, 61], which is also widely used, albeit less effective for high-dimensional input feature spaces.

Branch-and-bound verification is illustrated by Algorithm 1. The procedure maintains a list of unresolved robustness subproblems, initially containing the given robustness query augmented with an empty set of constraints (cf. Line 2). Each unresolved subproblem Q spawns a counterexample search (cf. Line 4). If the search is successful, a counterfactual is found and the procedure terminates immediately. Otherwise, the subproblem Q is given to the analyzer a in the *bounding step* (cf. Line 6). If $v(Q) \leq 0$, the verification is inconclusive and the algorithm performs the *branching step* (cf. Line 7): the unresolved subproblem Q is split into further subproblems Q_1 and Q_2 (cf. Line 8) which are added back to the list of unresolved subproblems (cf. Line 9). The splitting is done according to the partitioning strategy; in case of ReLU splitting, the branching refines the subproblem $Q = (\mathbf{x}, \mathcal{A}, \epsilon, f, C)$ by selecting a ReLU whose pre-activation value $\hat{x}_{i,j}$ is not yet constrained by C and partitioning the search space according to its possible sign status, yielding $Q_1 = (\mathbf{x}, \mathcal{A}, \epsilon, f, C \cup \{\hat{x}_{i,j} \geq 0\})$ and $Q_2 = (\mathbf{x}, \mathcal{A}, \epsilon, f, C \cup \{\hat{x}_{i,j} < 0\})$. The branch-and-bound process continues until all unresolved subproblems have been resolved, in which case robustness is verified (cf. Line 10).

Due to the NP-hardness of neural network verification [32], the number of subproblems generated by branch-and-bound can grow exponentially in the worst case. Therefore, practical implementations typically enforce a timeout, terminating the search and returning an inconclusive answer once the allocated verification time has elapsed (see Algorithm 2, where the differences with respect to Algorithm 1 are highlighted in boxes). An alternative strategy

Algorithm 2 Branch-and-Bound with Timeout

```
function BAB($a, (\mathbf{x}, \mathcal{A}, \epsilon, f), \boxed{T}$)
                                  ▷ $a\colon \hat{\mathcal{Q}} \to \mathbb{R}, (\mathbf{x}, \mathcal{A}, \epsilon, f) \in \mathcal{Q}, T \in \mathbb{N}$
    $\boxed{t_1 \leftarrow \text{TIME}()}$
    unresolved $\leftarrow \{(\mathbf{x}, \mathcal{A}, \epsilon, f, \emptyset)\}$      ▷ $(\mathbf{x}, \mathcal{A}, \epsilon, f, \emptyset) \in \hat{\mathcal{Q}}$
    for $Q \in$ unresolved do
        if CEX$(\mathbf{x}, \mathcal{A}, \epsilon, f)$ then return $-1$
        unresolved $\leftarrow$ unresolved $\setminus \{Q\}$
        result $\leftarrow a(Q)$                                  ▷ Bounding step
        if result $\leq 0$ then                                 ▷ Branching step
            $t_2 \leftarrow \text{TIME}()$
            if $t_2 - t_1 < T$ then
                $Q_1, Q_2 \leftarrow \text{SPLIT}(Q)$
                unresolved $\leftarrow$ unresolved $\cup \{Q_1, Q_2\}$
            else return 0
    return 1
```

is to cap the number of branching steps or the total processed subproblems [11].

3.2 Verified Explanations

Optimal Robust Explanations. Recent work [27, 42, 43] has established a tight connection between local robustness and explainability of machine learning classifiers. In particular, *abductive explanations* [26, 45], *prime implicants* [52], and *sufficient reasons* [8, 9] characterize the minimal subset of input features that are responsible for a classifier prediction, in the sense that any perturbation on the rest of the input features will never change the classification output. Restricting perturbations to a set of allowed perturbations leads to *distance-restricted* or *robust explanations* [34, 65, 25]:

▶ **Definition 1** (Robust Explanation). *Given a classifier $f\colon \mathbb{R}^d \to \mathbb{R}^k$, an input vector $\mathbf{x} \in \mathbb{R}^d$, and a perturbation radius $\epsilon > 0$, a* robust explanation *is a subset of the indexes of the input features $\mathcal{E}_\mathbf{x} \subseteq \{1, \dots, d\}$ such that f is locally robust on $\mathbf{x}$ to perturbations in $B^\epsilon_{\overline{\mathcal{E}}_\mathbf{x}}(\mathbf{x})$:*

$$\forall \mathbf{x}' \in B^\epsilon_{\overline{\mathcal{E}}_\mathbf{x}}(\mathbf{x})\colon y_f(\mathbf{x}') = y_f(\mathbf{x})$$

Owing to Definition 1, we will call $\overline{\mathcal{E}}_\mathbf{x}$ the *invariants* for $\mathbf{x}$. Note that, the full set of input features indexes $\{1, \dots, d\}$ is a trivially robust explanation. More generally, many subsets of feature indexes can constitute a robust explanation, but we are particularly interested in the subsets that contain no superfluous features [27, 65]:

▶ **Definition 2** (Optimal Explanation). *Given a classifier $f\colon \mathbb{R}^d \to \mathbb{R}^k$, an input vector $\mathbf{x} \in \mathbb{R}^d$, and a perturbation radius $\epsilon > 0$, a robust explanation $\mathcal{E}_\mathbf{x}$ is* optimal *if removing an index from $\mathcal{E}_\mathbf{x}$ would break local robustness:*

$$\forall\ i \in \mathcal{E}_\mathbf{x}\colon \exists\ \mathbf{x}^* \in B^\epsilon_{\overline{\mathcal{E}}_\mathbf{x} \cup \{i\}}(\mathbf{x})\colon y_f(\mathbf{x}^*) \neq y_f(\mathbf{x}) \qquad (1)$$

Note that, based on Definition 2, establishing the optimality of a robust explanation $\mathcal{E}_\mathbf{x}$ entails obtaining a *counterfactual* $\mathbf{x}^*$ witnessing Equation (1) for each individual feature in $\mathcal{E}_\mathbf{x}$.

Computing Robust Explanations. Existing algorithms for computing (optimal) robust explanations exploit a local robustness verifier as an *oracle* for identifying invariants or

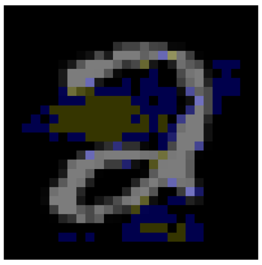


Figure 4 Example of v-optimal robust explanation.

finding counterfactuals. The simplest algorithms [65, 25] operate sequentially, mimicking the deletion-based algorithm [7] for computing minimal unsatisfiable subsets of logic formulas. Other [28, 64] implement dichotomic search [23] or adapt the quickXplain algorithm [31].

Despite these advances, scalability remains a challenge: computing optimal explanation is possible for neural network classifiers with 20-60 ReLUs [65, 64], while only robust but not necessarily optimal explanations can be computed for larger machine learning models with a large number of input features [65, 28]. In particular, in our experiments, we found that finding counterfactuals rapidly becomes infeasible. For instance, running our FaVeX algorithm for computing robust explanations with a 60-second timeout per robustness query on state-of-the-art models (with roughly $230k$ ReLUs) from the certified training literature [12] finds no counterexamples despite multiple hours of computation per image (cf. Table 3 in Section 7).

To bridge this gap between the *ideal* notion of optimal robust explanations (Definition 2) and its *practical realizability*, we introduce in Section 4 the concept of *verifier-optimal robust explanations*, which explicitly takes into account the underlying verifier, thereby capturing what can be provably achieved in practice. Importantly, this definition enables finding counterfactuals even at larger scale. Building on this notion, Section 5 presents an efficient algorithm for computing (verifier-optimal) robust explanations.

4 Verifier-Optimal Robust Explanations

The definition of optimal robust explanation (Definition 2) partitions the input feature indexes into the explanation $\mathcal{E}_{\mathbf{x}}$ and the invariants $\overline{\mathcal{E}}_{\mathbf{x}}$, implicitly requiring that the underlying verifier is complete, i.e., always able to verify or find a counterfactual for a given robustness query. This makes the notion of optimal robust explanation inherently semantic: it characterizes what explanations *should* be in principle, regardless of the limitations of practical verifiers.

While this semantic notion provides the ideal target, in practice verifiers are incomplete and may fail to decide a robustness query. To account for such uncertainty, we introduce a *practically achievable* counterpart of optimal robust explanations, that captures what can be established using a given verifier v. Specifically, our verifier-aware definition partitions the input feature indexes into three disjoint sets:

1. the invariants $\mathcal{R}_{\mathbf{x}}$ (not highlighted in Figure 4): the feature indexes that are surely not part of the explanation, for which the verifier v proves robustness (returns 1);
2. the counterfactuals $\mathcal{C}_{\mathbf{x}}$ (highlighted in yellow in Figure 4): the feature indexes that decisive for the explanations, for which the verifier v finds a counterfactual (returns -1);
3. the unknowns $\mathcal{U}_{\mathbf{x}}$ ((highlighted in blue in Figure 4): the feature indexes that remain ambiguous for the explanation, for which the verifier v remains inconclusive (returns 0);

Intuitively, $\mathcal{R}_{\mathbf{x}}$ corresponds to input features that do not affect the classification outcome, while the union $\mathcal{C}_{\mathbf{x}} \cup \mathcal{U}_{\mathbf{x}}$ of the counterfactuals and the unknowns yields a *hierarchical explanation* based on the ability of the given verifier v to find counterfactuals: it suggests a strong sensitivity of the classification output to perturbations of the features indexed by $\mathcal{C}_{\mathbf{x}}$, and a smaller degree of sensitivity to perturbations of the features indexed by $\mathcal{U}_{\mathbf{x}}$.

We formally define our verifier-optimal explanations below.

▶ **Definition 3** (Verifier-Optimal Robust Explanation). *Given a classifier $f : \mathbb{R}^d \to \mathbb{R}^k$, an input vector $\mathbf{x} \in \mathbb{R}^d$, a perturbation radius $\epsilon > 0$, and a verifier $v : \mathcal{Q} \to \{-1, 0, 1\}$, a* v-optimal robust explanation *is the union $\mathcal{C}_{\mathbf{x}} \cup \mathcal{U}_{\mathbf{x}}$ of disjoint subsets of the input feature indexes $\mathcal{C}_{\mathbf{x}} \subseteq \{1, \dots, d\}$ and $\mathcal{U}_{\mathbf{x}} \subseteq \{1, \dots, d\}$ such that:*

1. *$\mathcal{R}_{\mathbf{x}} \overset{def}{=} \overline{\mathcal{C}_{\mathbf{x}} \cup \mathcal{U}_{\mathbf{x}}}$ is the inclusion-maximal set of feature indexes such that the verifier v proves local robustness of f on $\mathbf{x}$ to perturbations in $B^{\epsilon}_{\mathcal{R}_{\mathbf{x}}}(\mathbf{x})$, i.e., $v((\mathbf{x}, \mathcal{R}_{\mathbf{x}}, \epsilon, f)) = 1$ and, for all $j \in \mathcal{C}_{\mathbf{x}} \cup \mathcal{U}_{\mathbf{x}}$, $v((\mathbf{x}, \mathcal{R}_{\mathbf{x}} \cup \{j\}, \epsilon, f)) \neq 1$;*
2. *v remains inconclusive when proving local robustness of f on $\mathbf{x}$ to perturbations in $B^{\epsilon}_{\mathcal{R}_{\mathbf{x}} \cup \mathcal{U}_{\mathbf{x}}}(\mathbf{x})$, i.e., $v((\mathbf{x}, \mathcal{R}_{\mathbf{x}} \cup \mathcal{U}_{\mathbf{x}}, \epsilon, f)) = 0$;*
3. *the verifier v always finds counterfactuals when proving local robustness of f on $\mathbf{x}$ when perturbations are allowed to all feature indexed by $\mathcal{R}_{\mathbf{x}} \cup \mathcal{U}_{\mathbf{x}}$ and* any *feature indexed by $\mathcal{C}_{\mathbf{x}}$, i.e., $\forall\, i \in \mathcal{C}_{\mathbf{x}} : v((\mathbf{x}, \mathcal{R}_{\mathbf{x}} \cup \mathcal{U}_{\mathbf{x}} \cup \{i\}, \epsilon, f)) = -1$.*

The first condition of Definition 3 implies that $\mathcal{U}_{\mathbf{x}} \cup \mathcal{C}_{\mathbf{x}}$ is a robust, yet in general not optimal, explanation. However, differently from Definitions 1 and 2, where perturbations are restricted to input features indexed by the invariants, in the third condition of Definition 3 perturbations are also allowed to all input features indexed by $\mathcal{U}_{\mathbf{x}}$. This increases the size of the perturbation space considered by the verifier, which in turn produces explanations that are *potentially larger* than those obtained when considering only perturbations to the invariants. On the other hand, we argue that these explanations are *more informative* since the input features indexed by $\mathcal{U}_{\mathbf{x}}$ do not produce counterfactuals individually, but counterfactuals found in combinations with these unknowns highlight clearly decisive input features for the classification outcome (the features indexed by $\mathcal{C}_{\mathbf{x}}$). In our experiments, we found that also allowing perturbations to all input features indexed by the unknown is crucial to enable finding counterfactuals even for larger scale classifiers. Note that, Definition 2 is an instance of Definition 3 in which the verifier is complete $\overline{v}$: $\mathcal{Q} \to \{-1, 1\}$:

▶ **Lemma 4.** *Given a classifier $f : \mathbb{R}^d \to \mathbb{R}^k$, an input vector $\mathbf{x} \in \mathbb{R}^d$, a perturbation radius $\epsilon > 0$, and a* complete *verifier $\overline{v} : \mathcal{Q} \to \{-1, 1\}$, a* v-optimal robust explanation *(Definition 3) is an optimal robust explanation (Definition 2).*

Proof. Since the verifier is *complete*, it never returns 0. Thus, the second condition of Definition 3 never occurs. Hence, $\mathcal{U}_{\mathbf{x}} = \emptyset$ and Definition 3 coincides with Definition 2. ◀

5 FaVeX: Faster Verified Explanations

In this section, we introduce FaVeX, our efficient algorithm for computing (verifier-optimal) robust explanations. We first present the main algorithm (Section 5.1) and then describe two acceleration strategies: an approach to speed up verification by reusing prior branches in branch-and-bound verification (Section 5.2), and a technique to speed up the search for counterfactuals by means of adversarial attacks in a restricted space (Section 5.3).

Algorithm 3 FaVeX

```
1: function FaVeX($\text{V-OPT}, f, \mathbf{x}, \epsilon, a, \vec{\mathcal{A}}, T$)    ▷ $\text{V-OPT} \in \{\text{TRUE}, \text{FALSE}\}$
       ▷ $f\colon \mathbb{R}^d \to \mathbb{R}^k$, $\mathbf{x} \in \mathbb{R}^d$, $\epsilon > 0$, $a\colon \hat{\mathcal{Q}} \to \mathbb{R}$, $\vec{\mathcal{A}} \in Sym(\{1,\dots,d\})$, $T \in \mathbb{N}$
2:     $\mathcal{R}_\mathbf{x}, \mathcal{U}_\mathbf{x}, \mathcal{C}_\mathbf{x} \leftarrow \emptyset, \emptyset, \emptyset$
3:     fallback ← FALSE
4:     batches ← $\{\vec{\mathcal{A}}\}$
5:     for $B \in$ batches do
6:         batches ← batches $\setminus \{B\}$
7:         if $|B| > 1$ then    ▷ batch robustness query
8:             if fallback then    ▷ fallback to sequential processing
9:                 for $i \in B$ do
10:                    if V-OPT then
11:                        result ← BaB($a, (\mathbf{x}, \mathcal{R}_\mathbf{x} \cup \mathcal{U}_\mathbf{x} \cup \{i\}, \epsilon, f), T$)
12:                    else result ← BaB($a, (\mathbf{x}, \mathcal{R}_\mathbf{x} \cup \{i\}, \epsilon, f), T$)
13:                    if result == 1 then $\mathcal{R}_\mathbf{x} \leftarrow \mathcal{R}_\mathbf{x} \cup \{i\}$
14:                    else if result = -1 then $\mathcal{C}_\mathbf{x} \leftarrow \mathcal{C}_\mathbf{x} \cup \{i\}$ else $\mathcal{U}_\mathbf{x} \leftarrow \mathcal{U}_\mathbf{x} \cup \{i\}$
15:            else    ▷ binary search-based batch processing
16:                if V-OPT then
17:                    result ← BaB($a, (\mathbf{x}, \mathcal{R}_\mathbf{x} \cup \mathcal{U}_\mathbf{x} \cup B, \epsilon, f), T/10$)
18:                else result ← BaB($a, (\mathbf{x}, \mathcal{R}_\mathbf{x} \cup B, \epsilon, f), T/10$)
19:                if result == 1 then $\mathcal{R}_\mathbf{x} \leftarrow \mathcal{R}_\mathbf{x} \cup B$
20:                else
21:                    $\vec{\mathcal{A}}_1, \vec{\mathcal{A}}_2 \leftarrow$ HALVE($\vec{\mathcal{A}}$)
22:                    batches ← batches $\cup \{\vec{\mathcal{A}}_1, \vec{\mathcal{A}}_2\}$
23:        else    ▷ single robustness query
24:            if V-OPT then
25:                result ← BaB($a, (\mathbf{x}, \mathcal{R}_\mathbf{x} \cup \mathcal{U}_\mathbf{x} \cup B, \epsilon, f), T$)
26:            else result ← BaB($a, (\mathbf{x}, \mathcal{R}_\mathbf{x} \cup B, \epsilon, f), T$)
27:            if result == 1 then $\mathcal{R}_\mathbf{x} \leftarrow \mathcal{R}_\mathbf{x} \cup B$
28:            else
29:                fallback ← TRUE
30:                if result = -1 then $\mathcal{C}_\mathbf{x} \leftarrow \mathcal{C}_\mathbf{x} \cup B$ else $\mathcal{U}_\mathbf{x} \leftarrow \mathcal{U}_\mathbf{x} \cup B$
31:    return $\mathcal{U}_\mathbf{x}$, $\mathcal{C}_\mathbf{x}$
```

5.1 Computing (Verifier-Optimal) Robust Explanations

FaVeX, shown in Algorithm 3, computes v-optimal robust explanations if the V-OPT flag (cf. Line 1) is TRUE, otherwise it computes robust explanations (cf. the parts highlighted in boxes in Algorithm 3). It maintains three sets $\mathcal{R}_\mathbf{x}$, $\mathcal{U}_\mathbf{x}$, and $\mathcal{C}_\mathbf{x}$, containing invariants, unknowns, and counterfactuals, all initially empty (cf. Line 2). It distinguishes between two kinds of robustness queries: *single queries*, in which perturbations are allowed to only one additional input feature besides the invariants in $\mathcal{R}_\mathbf{x}$ (cf. Lines 12 and 26) and, if the V-OPT flag is TRUE, the unknowns in $\mathcal{U}_\mathbf{x}$ (cf. Lines 11 and 25); and *batch queries*, in which perturbations are allowed to multiple additional features simultaneously (cf. Line 17 and 18). For batch queries, the timeout T enforced on the underlying chosen verifier v (cf. Line 1) is reduced by a factor of 10 (cf. Line 17 and 18). The reduction factor can be a parameter of the algorithm but, in our experiments, we found this value to offer the best trade-offs.

Similarly to [64], batch queries accelerate the identification of consecutive invariants

Algorithm 4 FaVeX Traversal Strategy

```
1: function FaVeX-traversal($f, \mathbf{x}, \epsilon, a$)
                                      ▷ $f : \mathbb{R}^d \to \mathbb{R}^k$, $\mathbf{x} \in \mathbb{R}^d$, $\epsilon > 0$, $a : \mathcal{Q} \to \mathbb{R}$
2:     $S \leftarrow (0, \ldots, 0)$                                  ▷ $S \in \mathbb{R}^d$
3:     for $i \in \{1, \ldots, d\}$ do
4:         $S_i \leftarrow a((\mathbf{x}, \{i\}, \epsilon, f))$
5:     return argsort($S$, descending)
```

by querying the verifier for multiple candidate input features at once, instead of making consecutive separate single queries for each of them. The search for batches of consecutive invariants is based on *binary search.* FaVeX takes as input a *traversal strategy* $\vec{\mathcal{A}}$, i.e., an ordered sequence of all the input feature indexes (cf. Line 1). The choice of traversal strategy matters for producing smaller explanations [65, 64]. We present our choice of traversal strategy later in this section. The first batch query allows perturbations to all input features indexed by $\vec{\mathcal{A}}$ (cf. Line 4). This query typically fails to verify (cf. Line 20), so $\vec{\mathcal{A}}$ is halved into two batches $\vec{\mathcal{A}}_1$ and $\vec{\mathcal{A}}_2$ (cf. Line 21) and binary search continues over them (cf. Line 22).

Once the batch size is down to one (cf. Line 23), batch queries become de facto single queries, and they trigger a fallback to sequential feature processing (cf. Line 29) if the verifier disproves local robustness of f on $\mathbf{x}$ or times out (cf. Line 28). Note that, unlike [64], we do not resume binary search over the next feature batches. In practice, we found that once batch queries have shrunk to size one and the verifier fails at this granularity, the likelihood of encountering further batches of consecutive invariants drops substantially. At this stage, continuing binary search adds overhead without providing benefits: it repeatedly queries increasingly small batches that are unlikely to yield new invariants, thereby worsening overall performance. For this reason, after the fallback is triggered (cf. Line 8), FaVeX switches entirely to sequential feature processing (cf. Lines 9-14).

The sets $\mathcal{R}_{\mathbf{x}}$, $\mathcal{U}_{\mathbf{x}}$, and $\mathcal{C}_{\mathbf{x}}$ are updated based on the result returned by the chosen verifier v: $\mathcal{R}_{\mathbf{x}}$ is grown if verification succeeds (v returns 1, cf. Lines 13, 19, and 27); otherwise, $\mathcal{C}_{\mathbf{x}}$ or $\mathcal{U}_{\mathbf{x}}$ are grown if counterfactuals are found (v returns -1) or if v times out (v returns 0), respectively (cf. Lines 14 and 30). Finally FaVeX returns $\mathcal{U}_{\mathbf{x}}$ and $\mathcal{C}_{\mathbf{x}}$ (cf. Line 31), whose union $\mathcal{U}_{\mathbf{x}} \cup \mathcal{C}_{\mathbf{x}}$ yields a v-optimal robust explanation (cf. Definition 3), if the v-opt flag is true, and a robust explanation (cf. Definition 1), otherwise.

Traversal Strategy. The traversal strategy used by FaVeX is shown in Algorithm 4. It leverages an analyzer a (cf. Line 1) to associate a *score* to each input feature index (cf. Line 3). The score is the lower bound of the worst-case logit difference computed by a, when the perturbations are restricted to the currently considered input feature index i (cf. Line 4). The input feature indexes are then sorted in descending order according to their score (cf. Line 5). The underlying intuition is that input feature indexes associated with a larger lower bound on the worst-case logit difference are more likely to be found invariants for the explanation by FaVeX. Note that our traversal strategy is similar to the one used in VeriX+ [64], with a subtle but important difference: the score associated to each input feature index by VeriX+ [64] is the lower bound on the logit of the true classification output instead of the worst-case logit difference. We argue that prioritizing indexes by their contribution to the worst-case logit difference is more faithful to the verification objective: it directly reflects the margin that must remain positive under perturbations, rather than focusing solely on the robustness of the true class logit without accounting for competing classes. This leads to a traversal strategy that better aligns with the decision criterion

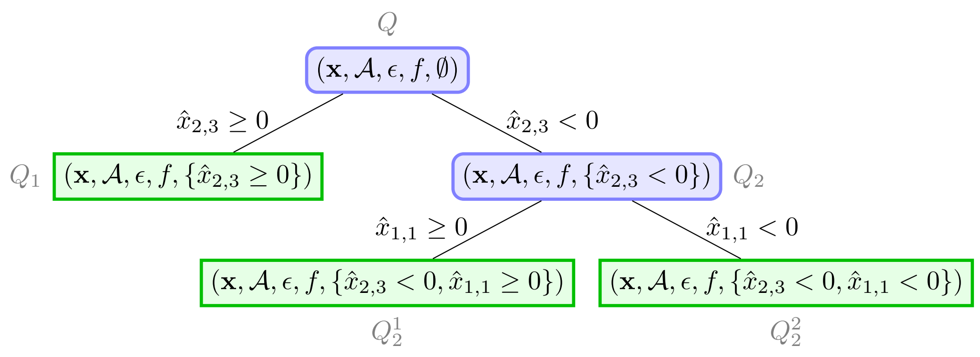

Q
$(\mathbf{x}, \mathcal{A}, \epsilon, f, \emptyset)$
$\hat{x}_{2,3} \geq 0$
$\hat{x}_{2,3} < 0$
Q_1
$(\mathbf{x}, \mathcal{A}, \epsilon, f, \{\hat{x}_{2,3} \geq 0\})$
$(\mathbf{x}, \mathcal{A}, \epsilon, f, \{\hat{x}_{2,3} < 0\})$
Q_2
$\hat{x}_{1,1} \geq 0$
$\hat{x}_{1,1} < 0$
$(\mathbf{x}, \mathcal{A}, \epsilon, f, \{\hat{x}_{2,3} < 0, \hat{x}_{1,1} \geq 0\})$
$(\mathbf{x}, \mathcal{A}, \epsilon, f, \{\hat{x}_{2,3} < 0, \hat{x}_{1,1} < 0\})$
Q_2^1
Q_2^2

Figure 5 Example of search tree explored by branch-and-bound.

used by the verifier and therefore improves the likelihood of identifying invariants early. An empirical comparison of different traversal strategies is shown in Tables 8-11 in Section 7.

5.2 Incremental Branch-and-Bound Verification

We can now remark that most often an invocation of the verifier v within FaVeX concerns a robustness query $(\mathbf{x}, \mathcal{A}, \epsilon, f)$ that differs only slightly from the previous one (typically $\mathcal{A}$ includes only one or, sometimes, few more input feature indexes). Instead of running v from scratch, we propose a strategy that exploits this incremental query evolution to reuse previous verifier computations and, in turn, speed up the overall procedure.

Specifically, we observed that branch-and-bound often performs the same sequence of branching steps for consecutive robustness queries. Thus, in some cases, we save and reuse the branching steps in later branch-and-bound calls. Concretely, inspired by IVAN [59], we cache the (constraints associated with the) robustness subproblems at the *leaves* of the search tree explored by branch-and-bound, and reuse them as starting point for subsequent branch-and-bound calls whenever applicable. Figure 5 shows an example of branch-and-bound search tree where the initial subproblem Q is first split into subproblems Q_1 and Q_2 (adding the constraints $\hat{x}_{2,3} \geq 0$ and $\hat{x}_{2,3} < 0$, respectively), and then Q_2 is further split into Q_2^1 and Q_2^2 (adding $\hat{x}_{1,1} \geq 0$ and $\hat{x}_{1,1} < 0$, respectively). In this case, we cache the (constraints associated with) subproblems Q_1, Q_2^1 and Q_2^2. The subsequent invocation of the verifier will reuse these subproblems (constraints) as a starting point for branch-and-bound, instead of starting from scratch with the robustness query (with an empty set of constraints).

Algorithm 5 shows how we augment branch-and-bound with save and reuse capabilities. When a subproblem Q is verified, we cache the set of constraints Q_C associated with Q (cf. Line 12). Otherwise, if a counterfactual is found (cf. Line 7) or the timeout is hit (cf. Line 18, we cache the constraints of all still unresolved subproblems (cf. Lines 8 and 19). The set of cached constraints is returned together with the verification result (cf. Lines 9, 20, 21) to be reused by subsequent calls to branch-and-bound (cf. Lines 3-4).

In practice, we do not apply this save-and-reuse strategy indiscriminately. While in general it substantially accelerates verification, in some cases, it can be detrimental, e.g., when the number of cached constraints sets becomes too large. To avoid these pitfalls, we enable reuse only under specific conditions and impose (a configurable) limit on the size of the cache. Our practical design choices and heuristics governing when to save and when to reuse constraints of branch-and-bound leaf subproblems are detailed in Section 6.2.

Algorithm 5 Branch-and-Bound (with Timeout and) Branching Save/Reuse

```
1: function BAB($a, (\mathbf{x}, \mathcal{A}, \epsilon, f), T, \mathbb{C}$)
                                  ▷ $a\colon \hat{\mathcal{Q}} \to \mathbb{R}, (\mathbf{x}, \mathcal{A}, \epsilon, f) \in \mathcal{Q}, T \in \mathbb{N}, \mathbb{C} \in \mathcal{P}(\mathcal{P}(\mathcal{C}))$
2:     $t_1 \leftarrow$ TIME()
3:     for $C \in \mathbb{C}$ do                                              ▷ Reuse
4:         unresolved $\leftarrow \{(\mathbf{x}, \mathcal{A}, \epsilon, f, C)\}$
5:     $\mathbb{C} \leftarrow \emptyset$
6:     for $Q \in$ unresolved do
7:         if CEX($\mathbf{x}, \mathcal{A}, \epsilon, f$) then
8:             for $Q' \in$ unresolved do $\mathbb{C} \leftarrow \mathbb{C} \cup \{Q'_C\}$      ▷ Save unresolved
9:             return $-1$, $\mathbb{C}$
10:        unresolved $\leftarrow$ unresolved $\setminus \{Q\}$
11:        result $\leftarrow a(Q)$
12:        if result $> 0$ then $\mathbb{C} \leftarrow \mathbb{C} \cup \{Q_C\}$                        ▷ Save $Q$
13:        else if result $\leq 0$ then
14:            $t_2 \leftarrow$ TIME()
15:            if $t_2 - t_1 < T$ then
16:                $Q_1, Q_2 \leftarrow$ SPLIT($Q$)
17:                unresolved $\leftarrow$ unresolved $\cup \{Q_1, Q_2\}$
18:            else
19:                for $Q' \in$ unresolved do $\mathbb{C} \leftarrow \mathbb{C} \cup \{Q'_C\}$  ▷ Save unresolved
20:                return 0, $\mathbb{C}$
21:    return 1, $\mathbb{C}$
```

5.3 Restricted-Space Counterfactual Search

The search for counterfactuals during branch-and-bound verification of a robustness query $(\mathbf{x}, \mathcal{A}, \epsilon, f)$ (cf., e.g., Line 4 in Algorithm 1) typically spans the entire set of allowed perturbations $B^{\epsilon}_{\mathcal{A}}(\mathbf{x})$. That is, all input features indexed by $\mathcal{A}$ are treated as freely perturbable, and thus the entire set $\mathcal{A}$ constitutes the counterfactuals *search space*.

However, we can again leverage the dependency between subsequent verification queries in FAVEX (as in Section 5.2), in this case to narrow the counterfactual search space, thereby reducing the search cost. When FAVEX operates in sequential query processing, in particular, input feature indexes are individually and iteratively added to set of feature indexes to which perturbations are allowed. If we have access to a counterfactual $\mathbf{x}' \in B^{\epsilon}_{\mathcal{A}}(\mathbf{x})$ for a robustness query $(\mathbf{x}, \mathcal{A}, \epsilon, f)$, it is extremely likely that x' will be very close to a counterfactual for the next robustness query $(\mathbf{x}, \mathcal{A} \cup B, \epsilon, f)$, in the larger set of allowed perturbations $B^{\epsilon}_{\mathcal{A}\cup B}(\mathbf{x})$.

This motivates our *restricted-space counterfactual search*. Concretely, when solving the next robustness query $(\mathbf{x}, \mathcal{A} \cup B, \epsilon, f)$, instead of conducting the counterfactual search over the entire search space $\mathcal{A} \cup B$ (left of Figure 6), we restrict it to only the newly-added input feature indexes in B (right of Figure 6). At the same time, we center the search space around the result $\mathbf{z} \in B^{\epsilon}_{\mathcal{A}}(\mathbf{x})$ of the previous counterfactual search, which is a point already known to be *at least close* to misclassification. That is, we fix the value of each input feature i indexed by $\mathcal{A}$ to $\mathbf{z}_i$. This way, the counterfactual search explores only a thin slice of $\mathcal{A} \cup B$, but one that is highly promising. In practice, we found this heuristic to be extremely effective for finding valid counterfactuals quickly. Implementation details are provided in Section 6.2.

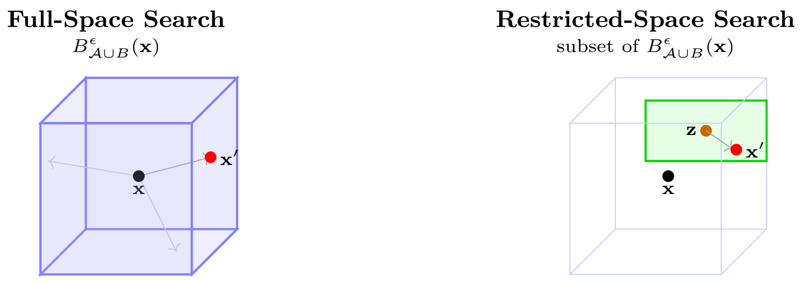

Full-Space Search
$B^{\epsilon}_{\mathcal{A}\cup B}(\mathbf{x})$
$\mathbf{x}$
$\mathbf{x}'$
Restricted-Space Search
subset of $B^{\epsilon}_{\mathcal{A}\cup B}(\mathbf{x})$
$\mathbf{z}$
$\mathbf{x}'$
$\mathbf{x}$

Figure 6 Illustration of full-space vs. restricted-space' counterfactual search. Full-space search (left) explores the entire perturbation region $B^{\epsilon}_{\mathcal{A}\cup B}(\mathbf{x})$. Restricted-space search (right) begins from the output $\mathbf{z}$ of the previous search (which may not be a counterfactual) and varies only the newly added feature(s), thus exploring a much smaller subset of $B^{\epsilon}_{\mathcal{A}\cup B}(\mathbf{x})$.

6 Implementation

Our implementation is based on the popular deep learning library PyTorch [47].

6.1 Verifier

We adopt the OVAL branch-and-bound framework [6, 4, 11, 10] as the backbone for our verifier, using a fixed timeout of either 300 seconds per query, or 60 seconds, depending on the benchmark. Note that while branch-and-bound is complete in principle, the timeout makes it incomplete in practice (cf. end of Section 3.1), as it is unlikely that the exponential worst-case runtime will be hit during the allotted timeout on non-trivial networks [32].

Bounding Phase. The bounding phase of branch-and-bound requires two components: one tries to prove robustness by operating on a network over-approximation. If the over-approximation is robust, then no counterexample can exist. This corresponds to computing a lower bound on the worst-case logit difference (cf. Section 3.1). We mainly rely on a popular dual-based algorithm, named α-β-CROWN [62], that solves a dual instance of a network relaxation that replaces the ReLU activations by over-approximating triangles [16]. This algorithm is both employed to bound the logit differences themselves, and to build each network pre-activation (a necessary preliminary step). In line with previous work, we only compute pre-activation bounds once at the branch-and-bound root (i.e., before any splitting) [11]. On smaller networks, we employ a framework configuration that may resort to a tighter network over-approximation for the bounds to the logit differences if needed [10]. We solve up to 2000 branch-and-bound sub-problems in parallel at any time, leveraging GPU acceleration through PyTorch. The other component, based on the evaluation of concrete points, seeks to find a counterexample violating robustness, upper bounding the worst-case logit difference. In practice, we first run a relatively inexpensive adversarial attack based on a local optimizer, a 10-step Projected Gradient Descent [41] (PGD-10) using the minimal logit difference as loss function, before entering the branch-and-bound loop. Within branch-and-bound, concrete points to evaluate are collected as a by-product of the over-approximations, and a more expensive local optimizer [13] (500-step MI-FGSM) is repeatedly run.

Branching Phase. In all cases, we use the ReLU splitting partitioning strategy, which operates by splitting ambiguous ReLUs (i.e., their pre-activation can take both negative and positive values for the considered input perturbations) into their two linear phases [5]. In particular, we use a recent strategy [12] that heuristically branches on the ReLU which is deemed to impact the most the tightness of the over-approximation employed during the bounding phase.

Selective MILP Calls. In order to speed up verification on smaller networks, we modified the OVAL framework to allow for the use of a Mixed Integer Linear Programming (MILP) solver whenever a subproblem has fewer than a given number of ambiguous ReLUs. In practice, we use this functionality in two different settings: on very small networks, we run the MILP solver at the root of branch-and-bound (i.e., before any splitting), if the quicker dual-based bounds fail to verify robustness. On harder networks, where this may not pay off, we call the MILP solver when no ambiguous neurons are left, in order to prevent slowdowns related to the inappropriate convergence of dual-based bounding. MILPs are solved using Gurobi, a commercial black-box solver [20].

6.2 FaVeX

Restricted-Space Attack. We run the novel restricted-space counterfactual search described in Section 5.3 only if the preliminary PGD-10 attack (see Section 6.1) fails to find counterexample. We execute several searches for counterexample (128) in parallel over the GPU, starting from different points in the restricted search space. Specifically, we include both the extrema of the search-space (the interval lower and upper bounds), and a series of starting points sampled uniformly within the interval. PGD-10 [41] is then run for each of these starting points, using the minimal logit difference as loss function.

Traversal. Our traversal strategy (cf. Section 5.1) requires as many calls to the analyzer a as the number of input features. In practice, we execute a number of these in parallel (all, on smaller networks) on a GPU. We consider two analyzers: α-CROWN [66], and the less expensive IBP [18, 17].

Leaf Reuse. We implemented the leaf reuse described in Section 5.2 through a direct modification of the OVAL branch-and-bound code. We store leaves as a list of branching decisions, which are then enforced after the computation of the pre-activation bounds at the root. In practice, we do not necessarily inherit the leaves from the previous branch-and-bound call. For efficiency reasons, we may discard leaves (i.e., start branch-and-bound from scratch) when too many of them are stored (more than a tunable threshold). Analogously, we discard leaves in case the previous call is a timeout and FaVeX is processing robustness queries sequentially (cf. Section 5), as inheriting them would likely cause further timeouts. Finally, in case of timeouts during the binary search batch processing mode, we inherit the leaves from the branch-and-bound call preceding the timeout.

7 Experimental Evaluation

This section presents an experimental evaluation of verifier-optimal robust explanations (Section 7.2), of FaVeX (Section 7.3), and of the associated traversal strategies (Section 7.4), preceded by a description of the experimental setting (Section 7.1).

7.1 Experimental Setting

We carry out experiments on computer vision datasets popular in the formal explanations literature [65, 64]. In particular, we consider MNIST [35], a black-and-white handwritten digit classification task; a 10-class version of GTSRB [57] taken from previous work [65], a traffic sign recognition task; and CIFAR-10 [33], a popular 10-class image classification benchmarks (examples of classes including “airplane”, “automobile”, “cat”, “dog”).

Table 1 Size of the network architectures employed in the experimental evaluation. For convolutional networks, the number of activations is computed for GTSRB and CIFAR-10 inputs.

Network	N. layers	N. activations	N. parameters
`FC-10x2`	3	2.00×10^1	3.10×10^4
`FC-50x2`	3	1.00×10^2	1.57×10^5
`CNN-3`	3	2.46×10^4	2.18×10^5
`CNN-7`	7	2.30×10^5	1.72×10^7

Network Architectures. In order to assess the scalability of FAVEX and of the presented definition, we consider 4 different feedforward ReLU networks of varying sizes, detailed in Table 1. The hardness of neural network verification mainly depends on the number of activations acting non-linearly. `FC-10x2` is a fully connected network with two hidden layers of width equal to 10. `FC-50x2` displays the same structure, but with hidden layers of width 50. Both networks have relatively few activations, and were trained using standard (SGD-based) network training and ℓ_1 regularization. `CNN-7` is a 7-layer convolutional network that is commonly employed in the certified training literature (networks trained for verified robustness), where it reaches state-of-the-art performance [12]: we use this to showcase scalability on networks relevant to the broader robust machine learning literature, and take the trained networks directly from previous work [12]. As `CNN-7` has 2.3×10^5 activations, we also train (using the same training scheme) a narrower 3-layer version, named `CNN-3`, which has 2.46×10^4 activations. `CNN-7` and `CNN-3` were trained to be robust against perturbations of $\epsilon = 0.2$ and $\epsilon = \frac{2}{255}$ on MNIST, and CIFAR-10, respectively.

Computational Setup. We run all the experiments on a single GPU, using 6 CPU cores and allocating 20GB of RAM. In order to run multiple experiments at once, we use different machines, but allocate each combination of network and dataset to a specific machine for consistency. We employed the following GPU models: Nvidia RTX 6000, Nvidia RTX 8000. All machines are equipped with the following CPU: AMD EPIC 7302.

Tuning. Before launching the experiments, we tested different configurations of the OVAL framework in order to minimize the overall runtime. We found it beneficial to avoid the custom branch-and-bound, and to call the MILP solver before any branching (see Section 6.1) on all the `FC-10x2` experiments, and on the `FC-50x2` MNIST experiments. The only hyper-parameter associated to FAVEX is k, the maximal number of leaves to be stored for reuse before discarding them. We tune $k \in \{0, 25, 100, 500, 5000, 50000\}$ on test images not employed for the evaluation. We leave the number of leaves unbounded if the tuning points to $k = 50000$: this is the case on the fully-connected network instances for which we do not directly resort to the MILP solver (`FC-50x2` on GTSRB and CIFAR-10).

7.2 Scalable Explanations

We here evaluate the scalability improvements associated to the definition of verifier-optimal robust explanations, presented in Section 4. Specifically, we use FAVEX and a fixed traversal order (α-FAVEX for CIFAR-10 and FAVEX-IBP for MNIST) to compute both (standard) robust explanations and `OVAL`-optimal robust explanations, using a per-query timeout of 60 seconds on `CNN-3` and `CNN-7`. In order to allow for a comparison on the number of found counterfactuals, we report $|\mathcal{C}_\mathbf{x}|$ and $|\mathcal{U}_\mathbf{x}|$ separately for standard robust explanations, instead of only reporting $|\mathcal{E}_\mathbf{x}|$ with $\mathcal{E}_\mathbf{x} := \mathcal{C}_\mathbf{x} \cup \mathcal{U}_\mathbf{x}$ as per the original definition [65] (cf. Definitions 1 and 2). Tables 2 and 3 show standard robust explanations are extremely expensive to compute, ranging from roughly 45 minutes per image on MNIST for `CNN-3`, to above 7 hours per image on CIFAR-10 for `CNN-7`. Note that, despite this computational effort, no counterexamples can be found. On the other hand, `OVAL`-optimal robust explanations are significantly faster to compute, and result in a significant number of counterfactuals, clearly showing the superior scalability of the proposed definition.

7.3 Computing Explanations Faster

We now turn to evaluating the efficacy of FaVeX when computing both standard robust explanations, and verifier-optimal robust explanations, again using a fixed traversal strategy (α-FaVeX for CIFAR-10 and GTSRB, FaVeX-IBP for MNIST). All considered algorithms to compute the explanations rely on the OVAL framework as verifier, using the same configuration (see Section 6.1). We compare against Sequential, which processes all the input features sequentially (cf. Section 5), mirroring what done in VeriX [65], albeit with a different verifier. We further compare against Binary Search, which exclusively uses the binary search-based batch processing, mirroring VeriX+ [64], again with a different verifier.

Standard Robust Explanations. Tables 4 and 5 show that FaVeX is significantly faster than both baselines on all considered settings on both `FC-10x2` and `FC-50x2`. Binary Search is consistently better than sequentially considering the input features on all benchmarks, cutting runtime almost by a third on MNIST for `FC-10x2`. FaVeX, however, reduces runtime even further, attaining speedup factors of up to 13× on `FC-10x2` and up to 16× on `FC-50x2`. Note that on `FC-10x2` and for MNIST on `FC-50x2`, where the MILP solver is called at the branch-and-bound root, no explicit OVAL branching is carried out (see Section 6.1), so leaf reuse is not employed. Consequently, the remarkable improvements in these settings are to be attributed to the reduced-space counterfactual search, which is catalyzed by the fact that FaVeX falls back to sequential robustness query processing.

Table 2 On `CNN-3`, `OVAL`-optimal robust explanations are significantly more scalable than standard robust explanations computed using the same verifier. Results are averaged over the first 10 images of the test set. Bold entries correspond to the smallest runtime, and the largest number of found counterfactuals.

	MNIST, $\epsilon = 0.25$			CIFAR-10, $\epsilon = \frac{8}{255}$		
Explanation	$\lvert\mathcal{C}_\mathbf{x}\rvert$	$\lvert\mathcal{U}_\mathbf{x}\rvert$	time [s]	$\lvert\mathcal{C}_\mathbf{x}\rvert$	$\lvert\mathcal{U}_\mathbf{x}\rvert$	time [s]
Standard robust explanation	0.00	247.80	2689.77	0.00	461.00	8984.10
V-optimal robust explanation	**160.30**	94.40	**612.75**	**210.40**	251.70	**1150.82**

Table 3 `OVAL`-optimal robust explanations are significantly more scalable than standard robust explanations computed using the same verifier on `CNN-7`. We average results over the first 3 images of the test set, and highlight in bold entries correspond to the smallest runtime and to the largest number of found counterfactuals.

	MNIST, $\epsilon = 0.25$			CIFAR-10, $\epsilon = \frac{16}{255}$		
Explanation	$\lvert\mathcal{C}_\mathbf{x}\rvert$	$\lvert\mathcal{U}_\mathbf{x}\rvert$	time [s]	$\lvert\mathcal{C}_\mathbf{x}\rvert$	$\lvert\mathcal{U}_\mathbf{x}\rvert$	time [s]
Standard robust explanation	0.00	452.00	14 317.57	0.00	730.67	25 487.29
V-optimal robust explanation	**207.33**	249.33	**4446.53**	**467.00**	266.33	**6532.98**

Table 4 FaVeX attains significant speed-ups against previous algorithms to compute standard robust explanations [65, 64] on `FC-10x2`. Results are averaged over the first 100 images of the test set. Bold entries correspond to the smallest runtime.

	MNIST, $\epsilon = 0.1$		GTSRB, $\epsilon = 0.05$	
Method	$\lvert\mathcal{E}_{\mathbf{x}}\rvert$	time [s]	$\lvert\mathcal{E}_{\mathbf{x}}\rvert$	time [s]
Sequential	70.90	182.62	352.26	518.34
Binary Search	70.90	65.21	352.26	345.48
FaVeX	70.90	**21.21**	352.26	**26.08**

Table 5 FaVeX is significantly faster than previous algorithms to compute standard robust explanations [65, 64] on `FC-50x2`. Results are averaged over the first 100 images of the test set. Bold entries correspond to the smallest runtime.

	MNIST, $\epsilon = 0.2$		GTSRB, $\epsilon = 0.1$		CIFAR-10, $\epsilon = \frac{2}{255}$	
Method	$\lvert\mathcal{E}_{\mathbf{x}}\rvert$	time [s]	$\lvert\mathcal{E}_{\mathbf{x}}\rvert$	time [s]	$\lvert\mathcal{E}_{\mathbf{x}}\rvert$	time [s]
Sequential	82.74	243.38	287.02	688.24	204.23	468.23
Binary Search	82.74	146.45	287.02	476.55	204.23	197.55
FaVeX	82.74	**48.16**	287.02	**30.48**	204.23	**13.92**

Table 6 On `CNN-3`, FaVeX computes `OVAL`-optimal robust explanations faster than previous computation strategies [65, 64] while at the same time finding more counter-factuals. Results are averaged over the first 10 images of the test set. Bold entries correspond to the smallest runtime and the largest number of provided counterfactuals.

	MNIST, $\epsilon = 0.25$			CIFAR-10, $\epsilon = \frac{8}{255}$		
Method	$\lvert\mathcal{C}_{\mathbf{x}}\rvert$	$\lvert\mathcal{U}_{\mathbf{x}}\rvert$	time [s]	$\lvert\mathcal{C}_{\mathbf{x}}\rvert$	$\lvert\mathcal{U}_{\mathbf{x}}\rvert$	time [s]
Sequential	129.10	125.50	1164.89	209.30	253.10	1287.96
Binary Search	134.20	120.40	1026.67	209.60	252.70	1367.25
FaVeX	**160.30**	94.40	**612.75**	**210.40**	251.70	**1150.82**

Table 7 On `CNN-7`, FaVeX computes `OVAL`-optimal robust explanations faster than previous computation strategies [65, 64] while at the same time finding more counter-factuals. Results are averaged over the first 3 images of the test set. Bold entries correspond to the smallest runtime and the largest number of provided counterfactuals.

	MNIST, $\epsilon = 0.25$			CIFAR-10, $\epsilon = \frac{16}{255}$		
Method	$\lvert\mathcal{C}_{\mathbf{x}}\rvert$	$\lvert\mathcal{U}_{\mathbf{x}}\rvert$	time [s]	$\lvert\mathcal{C}_{\mathbf{x}}\rvert$	$\lvert\mathcal{U}_{\mathbf{x}}\rvert$	time [s]
Sequential	142.33	315.00	7979.61	464.00	269.33	6868.53
Binary Search	148.33	308.67	9165.06	464.33	269.00	8284.12
FaVeX	**207.33**	249.33	**4446.53**	**467.00**	266.33	**6532.98**

Verifier-Optimal Robust Explanations. As visible from Tables 6 and 7, FaVeX is also beneficial when computing `OVAL`-optimal robust explanations on larger networks, with speed-ups up to 67% and 79% on MNIST for `CNN-3` and for `CNN-7`, respectively. Remarkably, the use of FaVeX also results in a larger number of counterfactuals being found, pointing to its superior efficacy in terms of counterfactual search. Interestingly, Sequential is almost

Table 8 Comparison of different traversal strategies for standard robust explanations on `FC-10x2`. Results are averaged over the first 100 images of the test set. Bold entries correspond to the smallest explanation.

	MNIST, $\epsilon = 0.1$		GTSRB, $\epsilon = 0.05$	
Traversal	$\lvert\mathcal{E}_\mathbf{x}\rvert$	time [s]	$\lvert\mathcal{E}_\mathbf{x}\rvert$	time [s]
VeriX	91.26	38.10	737.02	37.49
VeriX+	72.08	23.05	357.26	24.06
α-FaVeX	79.85	30.24	**352.26**	26.08
FaVeX-IBP	**70.90**	21.21	355.52	24.32

Table 9 Comparison of different traversal strategies for standard robust explanations on `FC-50x2`. Results are averaged over the first 100 images of the test set. Bold entries correspond to the smallest explanation.

	MNIST, $\epsilon = 0.2$		GTSRB, $\epsilon = 0.1$		CIFAR-10, $\epsilon = \frac{2}{255}$	
Traversal	$\lvert\mathcal{E}_\mathbf{x}\rvert$	time [s]	$\lvert\mathcal{E}_\mathbf{x}\rvert$	time [s]	$\lvert\mathcal{E}_\mathbf{x}\rvert$	time [s]
VeriX	94.02	85.00	851.64	117.20	310.56	18.40
VeriX+	**77.46**	48.17	289.56	25.74	222.66	13.85
α-FaVeX	102.35	67.01	**287.02**	30.48	**204.23**	13.92
FaVeX-IBP	82.74	48.16	287.25	25.31	220.96	14.02

always preferable to Binary Search on these networks: this is due to the relatively large size of the explanations. In fact, for larger explanations, Binary Search will go relatively deep in most branches of its recursion tree, hence resulting in a larger number of calls to the verifier. Finally, it is worth pointing out that the time to compute the explanations sharply increases with the size of the network. Focusing on MNIST, this takes FaVeX 21.21 seconds on average for `FC-10x2`, 48.16 on `FC-50x2`, 612.75 seconds on `CNN-3`, and 4446.53 seconds on `CNN-3`.

7.4 Traversal Analysis

We now turn our attention to studying the effect of the employed traversal strategies, comparing α-FaVeX and FaVeX-IBP with the VeriX [65] and VeriX+ [64] traversals. We use FaVeX to compute the explanations owing to its superior speed.

Standard Robust Explanations. Tables 8 and 9 show the results of the analysis for standard robust explanations on smaller networks (`FC-10x2` and `FC-50x2`). While α-FaVeX consistently yields the smallest explanations on GTSRB and CIFAR-10, it underperforms on MNIST, where strategies based on IBP attain superior performance. Nevertheless, on GTSRB and CIFAR-10, except for the VeriX strategy, which appears to produce markedly larger explanations, the performance of the considered strategies is not particularly dissimilar, with the best size reduction of α-FaVeX (attained on CIFAR-10 for `FC-50x2`) compared to VeriX+ being 9%. Finally, we remark that `FC-50x2` appears to be more robust than the smaller `FC-10x2`: on GTSRB, smaller explanations are produced by all traversals except VeriX, in spite of the twice-larger considered perturbation radius.

Verifier-Optimal Robust Explanations. Tables 10 and 11 study traversals for OVAL-optimal robust explanations on the larger `CNN-3` and `CNN-7`. In order to allow for a coherent comparison, we here consider the explanation as the union of $\mathcal{C}_\mathbf{x}$ and $\mathcal{U}_\mathbf{x}$. As for the smaller networks, VeriX+ produces smaller explanations on MNIST. On the other hand, the margins of improvement of α-FaVeX are reduced, with FaVeX-IBP producing marginally smaller explanations on CIFAR-10 for `CNN-7`. At the same time, α-FaVeX non-negligibly reduces the number of counterfactuals on CIFAR-10 for both networks, highlighting that the allocation of pixels between $\mathcal{C}_\mathbf{x}$ and $\mathcal{U}_\mathbf{x}$ depends on the traversal strategy.

Table 10 Comparison of different traversal strategies for `OVAL`-optimal robust explanations on `CNN-3`. Results are averaged over the first 10 images of the test set. Bold entries correspond to the smallest $\mathcal{C}_\mathbf{x} \cup \mathcal{U}_\mathbf{x}$.

	MNIST, $\epsilon = 0.25$				CIFAR-10, $\epsilon = \frac{8}{255}$			
Method	$\lvert\mathcal{C}_\mathbf{x} \cup \mathcal{U}_\mathbf{x}\rvert$	$\lvert\mathcal{C}_\mathbf{x}\rvert$	$\lvert\mathcal{U}_\mathbf{x}\rvert$	time [s]	$\lvert\mathcal{C}_\mathbf{x} \cup \mathcal{U}_\mathbf{x}\rvert$	$\lvert\mathcal{C}_\mathbf{x}\rvert$	$\lvert\mathcal{U}_\mathbf{x}\rvert$	time [s]
VeriX	282.80	160.50	122.30	974.35	765.50	437.20	328.30	1925.09
VeriX+	**247.40**	155.20	92.20	576.67	468.90	262.00	206.90	915.45
α-FaVeX	289.90	181.20	108.70	696.31	**462.10**	210.40	251.70	1150.82
FaVeX-IBP	254.70	160.30	94.40	612.75	466.40	250.90	215.50	941.95

Table 11 Comparison of different traversal strategies for `OVAL`-optimal robust explanations on `CNN-7`. Results are averaged over the first 3 images of the test set. Bold entries correspond to the smallest $\mathcal{C}_\mathbf{x} \cup \mathcal{U}_\mathbf{x}$.

	MNIST, $\epsilon = 0.25$				CIFAR-10, $\epsilon = \frac{16}{255}$			
Method	$\lvert\mathcal{C}_\mathbf{x} \cup \mathcal{U}_\mathbf{x}\rvert$	$\lvert\mathcal{C}_\mathbf{x}\rvert$	$\lvert\mathcal{U}_\mathbf{x}\rvert$	time [s]	$\lvert\mathcal{C}_\mathbf{x} \cup \mathcal{U}_\mathbf{x}\rvert$	$\lvert\mathcal{C}_\mathbf{x}\rvert$	$\lvert\mathcal{U}_\mathbf{x}\rvert$	time [s]
VeriX	547.00	123.33	423.67	7763.91	945.00	728.67	216.33	4974.69
VeriX+	**429.33**	196.67	232.67	4394.67	735.67	522.00	213.67	5105.86
α-FaVeX	552.33	234.67	317.67	5344.59	733.33	467.00	266.33	6532.98
FaVeX-IBP	456.67	207.33	249.33	4446.53	**729.33**	512.67	216.67	5099.98

8 Related Work

A large body of work from the machine learning community has sought to provide explanations on the decisions of machine learning models. Earlier works from the machine learning community, for instance, propose to build interpretable approximations of the original model [40, 50, 48]. Another line of work tries to find points close to the input such that the decision would change (counterfactuals), which are robust in either the input space [36], or the parameter space [21, 29]. For a survey on robust counterfactual explanation in machine learning, we direct the reader to [30]. At the same time, multiple approaches to heuristically select a subset of features deemed responsible for the predictions exist, typically based on model gradients [53, 51]. However, these heuristics lack any type of formal guarantees, which are however crucial for safety-critical systems. Earlier work on formal explainability was carried out by the logic community, focusing on abductive explanations [44, 42, 45] (cf. Section 3.2). This was followed by work that adds locality constraints, necessary to obtain meaningful explanations on complex models such as neural networks, first in natural language processing [34], and then in computer vision, where the computation of formal explanations has been defined through a series of robustness queries to a neural network verifier [65, 64].

Earlier approaches to neural network verification typically relied on the use of black-box solvers, either from the model checking community [32], or from mathematical optimization [58, 38]. All these complete approaches can be seen as specific instances of branch-and-bound [6]. Incomplete approaches, instead, rely on network over-approximations, and were derived in parallel by both the formal methods [37, 17, 56, 55, 61, 46], and the machine learning [18, 68] communities, often relying on tools from optimization [63, 15]. A series of incomplete verifiers has been designed and integrated within branch-and-bound (as bounding algorithm), yielding verifiers that scale significantly better than earlier approaches based on black-box solvers [4, 66, 12, 24]. The current state-of-the-art, as highlighted by yearly competitions in the area [3], relies on fast incomplete algorithms based on linear bound propagation techniques [62], possibly with the addition of optimizations such as custom cutting planes [67, 70]. A recent work [59], IVAN, proposes to reuse information from previous branch-and-bound calls when verifying slightly modified versions of the same network. Our incremental approach, presented in Section 5.2 is a simplified version of this idea yet applied to formal explainability, where the network stays the same, but the verifier is called on a series of similar input domains that differ in the perturbed features of the same input point.

9 Conclusion and Future Work

While verified explanations offer a theoretically-principled approach for explainable machine learning, their applicability is severely limited by their scalability. This work makes a step towards improving the scalability of verified explanations for neural networks classifiers by both introducing a novel algorithm for their computation, and presenting a novel and more realistic definition of verified explanations.

We proposed FaVeX (Section 5), a new algorithm that substantially accelerates the computation of verified explanations. Compared to previous work, FaVeX presents three key improvements:

(i) it dynamically combines both batch processing of robustness queries in a binary-search fashion and their sequential processing, getting the best of both worlds depending on the query (Section 5.1);

(ii) it reuses information from previous executions of branch-and-bound, thus accelerating the overall verification process (Section 5.2);

(iii) it leverages a novel reduced-space counterfactual search that prevents calls to branch-and-bound by effectively re-using information from previous queries, which are used to quickly find counterfactuals (Section 5.3).

Furthermore, we introduced *verifier-optimal robust explanations* (Definition 3), which explicitly account for the practical incompleteness of neural network verifiers on medium-sized networks by allowing features whose robustness cannot be practically ascertained to be perturbed along the invariants, and produce a hierarchical explanation. In the case of a perfect verifier, the presented definition reverts to the standard definition of an optimal robust explanation from previous work (Lemma 4).

Our experiments demonstrate that FaVeX yields speed-ups of up to an order of magnitude compared to previous algorithms when computing standard robust explanations on small fully-connected networks, and that it cuts explanation times while discovering more counterfactuals when computing verifier-optimal robust explanations on larger convolutional networks, At the same time we show that, using FaVeX, verifier-optimal robust explanations can be computed for a fraction of the cost of the standard definition of robust explanations, while at the same time allowing for the production of several counterfactuals, which cannot be otherwise provided. In particular, FaVeX can find hundreds of counterfactuals on state-of-the-art networks trained for provable robustness to small adversarial examples, which feature hundreds of thousands of neurons, hence producing formal explanations at scale.

Future Work. While this work substantially improves on the scalability of formal explanations, their computation still requires in the order of an hour per image on networks with tens of millions of parameters, which are very small compared to real-world applications in many domains. Further progress on scalability is hence fundamental to ensure their widest applicability. Promising directions in this sense include: exploring more complex ways to re-use information from the various queries to the verifier, exploring the interaction between training methods and formal explanability, and devising ad-hoc algorithms to train networks that combine good performance and formal explanability. Finally, while our experiments focus on vision models, testing and potentially adapting verifier-optimal robust explanations and FaVeX to other data domains is an exciting avenue for future work.

References

1 Aws Albarghouthi. Introduction to neural network verification. *CoRR*, abs/2109.10317, 2021. URL: https://arxiv.org/abs/2109.10317.

2 Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Math. Program.*, 183(1):3–39, 2020.

3 Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T Johnson, and Changliu Liu. First three years of the international verification of neural networks competition (vnn-comp). *International Journal on Software Tools for Technology Transfer*, 25(3):329–339, 2023.

4 Rudy Bunel, Alessandro De Palma, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip HS Torr, and M Pawan Kumar. Lagrangian decomposition for neural network verification. In *Conference on Uncertainty in Artificial Intelligence*, 2020.

5 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. *J. Mach. Learn. Res.*, 21:42:1–42:39, 2020. URL: https://jmlr.org/papers/v21/19-468.html.

6 Rudy Bunel, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. A unified view of piecewise linear neural network verification. In *Neural Information Processing Systems*, 2018.

7 John W. Chinneck and Erik W. Dravnieks. Locating minimal infeasible constraint sets in linear programs. *INFORMS J. Comput.*, 3(2):157–168, 1991.

8 Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 712–720. IOS Press, 2020.

9 Adnan Darwiche and Auguste Hirth. On the (complete) reasons behind decisions. *J. Log. Lang. Inf.*, 32(1):63–88, 2023.

10 Alessandro De Palma, Harkirat Singh Behl, Rudy Bunel, Philip H. S. Torr, and M. Pawan Kumar. Scaling the convex barrier with sparse dual algorithms. *Journal of Machine Learning Research*, 2024.

11 Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr, and M. Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition. *CoRR*, abs/2104.06718, 2021. URL: https://arxiv.org/abs/2104.06718, `arXiv:2104.06718`.

12 Alessandro De Palma, Rudy Bunel, Krishnamurthy (Dj) Dvijotham, M. Pawan Kumar, Robert Stanforth, and Alessio Lomuscio. Expressive losses for verified robustness via convex combinations. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

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

14 Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

15 Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *Conference on Uncertainty in Artificial Intelligence*, 2018.

16 Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. *Automated Technology for Verification and Analysis*, 2017.

17 Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI2: safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 3–18. IEEE Computer Society, 2018.

18 Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Arthur Mann, and Pushmeet Kohli. Scalable verified training for provably robust image classification. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 4841–4850. IEEE, 2019.

19 Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5):93:1–93:42, 2019. `doi:10.1145/3236009`.

20 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. URL: https://www.gurobi.com.

21 Faisal Hamman, Erfaun Noorani, Saumitra Mishra, Daniele Magazzeni, and Sanghamitra Dutta. Robust counterfactual explanations for neural networks with probabilistic guarantees. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 12351–12367. PMLR, 23–29 Jul 2023. URL: https://proceedings.mlr.press/v202/hamman23a.html.

22 Yotam Hechtlinger. Interpretation of Prediction Models Using the Input Gradient. *CoRR arXiv*, 2016. URL: http://arxiv.org/abs/1611.07634.

23 Fred Hemery, Christophe Lecoutre, Lakhdar Sais, and Frédéric Boussemart. Extracting mucs from constraint networks. In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 113–117. IOS Press, 2006.

24 Patrick Henriksen and Alessio Lomuscio. DEEPSPLIT: an efficient splitting method for neural network verification via indirect effect analysis. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2549–2555. ijcai.org, 2021.

25 Xuanxiang Huang and João Marques-Silva. On the failings of shapley values for explainability. *Int. J. Approx. Reason.*, 171:109112, 2024.

26 Alexey Ignatiev, Nina Narodytska, and João Marques-Silva. Abduction-based explanations for machine learning models. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence,* *EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 1511–1519. AAAI Press, 2019.

27 Alexey Ignatiev, Nina Narodytska, and João Marques-Silva. On relating explanations and adversarial examples. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 15857–15867, 2019.

28 Yacine Izza, Xuanxiang Huang, António Morgado, Jordi Planes, Alexey Ignatiev, and João Marques-Silva. Distance-restricted explanations: Theoretical underpinnings & efficient implementation. In Pierre Marquis, Magdalena Ortiz, and Maurice Pagnucco, editors, *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024*, 2024.

29 Junqi Jiang, Jianglin Lan, Francesco Leofante, Antonio Rago, and Francesca Toni. Provably robust and plausible counterfactual explanations for neural networks via robust optimisation. In Berrin Yanıkoğlu and Wray Buntine, editors, *Proceedings of the 15th Asian Conference on Machine Learning*, volume 222 of *Proceedings of Machine Learning Research*, pages 582–597. PMLR, 11–14 Nov 2024. URL: https://proceedings.mlr.press/v222/jiang24a.html.

30 Junqi Jiang, Francesco Leofante, Antonio Rago, and Francesca Toni. Robust counterfactual explanations in machine learning: A survey. *CoRR*, abs/2402.01928, 2024. URL: https://doi.org/10.48550/arXiv.2402.01928, doi:10.48550/ARXIV.2402.01928.

31 Ulrich Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 167–172. AAAI Press / The MIT Press, 2004.

32 Guy Katz, Clark W. 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 Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 97–117. Springer, 2017. doi:10.1007/978-3-319-63387-9_5.

33 A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

34 Emanuele La Malfa, Rhiannon Michelmore, Agnieszka M. Zbrzezny, Nicola Paoletti, and Marta Kwiatkowska. On guaranteed optimal robust explanations for NLP models. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2658–2665. ijcai.org, 2021.

35 Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

36 Francesco Leofante and Nico Potyka. Promoting counterfactual robustness through diversity. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. doi:10.1609/aaai.v38i19.30127.

37 Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In Bor-Yuh Evan Chang, editor, *Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11822 of *Lecture Notes in Computer Science*, pages 296–319. Springer, 2019.

38 Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

39 Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 4765–4774, 2017.

40 Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Neural Information Processing Systems*, 2017.

41 Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

42 João Marques-Silva. Logic-based explainability in machine learning. In Leopoldo E. Bertossi and Guohui Xiao, editors, *Reasoning Web. Causality, Explanations and Declarative Knowledge - 18th International Summer School 2022, Berlin, Germany, September 27-30, 2022, Tutorial Lectures*, volume 13759 of *Lecture Notes in Computer Science*, pages 24–104. Springer, 2022.

43 João Marques-Silva. Logic-based explainability: Past, present and future. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Software Engineering Methodologies - 12th International Symposium, ISoLA 2024, Crete, Greece, October 27-31, 2024, Proceedings, Part IV*, volume 15222 of *Lecture Notes in Computer Science*, pages 181–204. Springer, 2024.

44 Joao Marques-Silva, Thomas Gerspacher, Martin C Cooper, Alexey Ignatiev, and Nina Narodytska. Explanations for monotonic classifiers. In Marina Meila and Tong Zhang, editors, *International Conference on Machine Learning*. PMLR, 2021.

45 João Marques-Silva and Alexey Ignatiev. Delivering trustworthy AI through formal XAI. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 12342–12350. AAAI Press, 2022.

46 Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin T. Vechev. PRIMA: general and precise neural network certification via scalable convex hull approximations. *Proc. ACM Program. Lang.*, 6(POPL):1–33, 2022.

47 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019.

48 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

49 Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. “Why should I trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016*, pages 1135–1144. ACM, 2016.

50 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

51 Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *IEEE International Conference on Computer Vision*, 2017.

52 Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 5103–5111. ijcai.org, 2018. URL: https://doi.org/10.24963/ijcai.2018/708, `doi:10.24963/IJCAI.2018/708`.

53 Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

54 Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin T. Vechev. Beyond the single neuron convex barrier for neural network certification. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 15072–15083, 2019.

55 Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. Fast and effective robustness certification. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10825–10836, 2018.

56 Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL):41:1–41:30, 2019.

57 Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.

58 Vincent Tjeng, Kai Yuanqing Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HyGIdiRqtm.

59 Shubham Ugare, Debangshu Banerjee, Sasa Misailovic, and Gagandeep Singh. Incremental verification of neural networks. *Proc. ACM Program. Lang.*, 7(PLDI):1920–1945, 2023. `doi:10.1145/3591299`.

60 Caterina Urban and Antoine Miné. A review of formal methods applied to machine learning. *CoRR*, abs/2104.02466, 2021. URL: https://arxiv.org/abs/2104.02466.

61 Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6369–6379, 2018.

62 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. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 29909–29921, 2021.

63 Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, 2018.

64 Min Wu, Xiaofu Li, Haoze Wu, and Clark W. Barrett. Better verified explanations with applications to incorrectness and out-of-distribution detection. *CoRR*, abs/2409.03060, 2024. `arXiv:2409.03060`, `doi:10.48550/ARXIV.2409.03060`.

65 Min Wu, Haoze Wu, and Clark W. Barrett. Verix: Towards verified explainability of deep neural networks. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

66 Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively

parallel incomplete verifiers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

67 Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-propagation-based neural network verification. In *Neural Information Processing Systems*, volume 35, 2022.

68 Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 4944–4953, 2018.

69 Duo Zhou, Christopher Brix, Grani A. Hanasusanto, and Huan Zhang. Scalable neural network verification with branch-and-bound inferred cutting planes. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

70 Duo Zhou, Christopher Brix, Grani A Hanasusanto, and Huan Zhang. Scalable neural network verification with branch-and-bound inferred cutting planes. In *Neural Information Processing Systems*, pages 29324–29353, 2024.