Efficient Complete Verification of Neural Networks via Layer-wised Splitting and Refinement

Banghu Yin, Liqian Chen, Jiangchao Liu, Ji Wang, *Member, IEEE*

***Abstract*—Safety and robustness properties are highly required for neural networks deployed in safety critical applications. Current complete verification techniques of these properties suffer from the lack of efficiency and effectiveness. In this paper, we present an efficient complete approach to verifying safety and robustness properties of neural networks through incrementally determinizing activation states of neurons. The key idea is to generate constraints via layer-wised splitting that make activation states of hidden neurons become deterministic efficiently, and which are then utilized for refining inputs systematically so that abstract analysis over the refined input can be more precise. Our approach decomposes a verification problem into a set of sub-problems via layer-wised input space splitting. The property is then checked in each sub-problem, where the activation states of at least one hidden neurons will be determinized. Further checking is accelerated by constraint-guided input refinement. We have implemented a parallel tool called LayerSAR to verify safety and robustness properties of ReLU neural networks in a sound and complete way, and evaluated it extensively on several benchmark sets. Experimental results show that our approach is promising, compared with complete tools such as Planet, Neurify, Marabou, ERAN, Venus, Venus2 and nnenum in verifying safety and robustness properties on the benchmarks.**

***Index Terms*—neural network verification, abstract analysis, input splitting, input refinement, complete verification**

I. Introduction

Neural networks have been widely used in safety-critical areas, such as autonomous driving [1], [2], medical diagnosis [3], and aircraft collision avoidance systems [4]. In such applications, any violations of critical properties can result in serious consequences even disasters. Therefore, ensuring the safety and robustness of such systems has become an important prerequisite for the deployment of neural network based techniques in real world. The *verification problem* is defined as follows [5]: given a neural network as a function with certain input-output relationship, to check the property that if the input belongs to some set of $\mathcal{X}$ then the output will belong to some set $\mathcal{Y}$.

Test and simulation are the mainstream techniques to validate the properties of neural networks at the moment. But due to the inherent weakness of these point-wise techniques, formal verification is highly desired for its provable guarantees in safety-critical applications [6]–[8]. The existing formal approaches can be classified into three main categories, i.e. reachability, optimization and search [5], [9]. The reachability approaches for verifying neural networks is to calculate the reachable output set with a given input set, and then check whether the property is satisfied under this reachable set [5]. Due to high-dimension of input, the nonlinearity of activation function and the large number of neurons contained in the neural network, it is prohibitively expensive to compute the concrete reachable set accurately. One natural solution is to use abstraction techniques to over-approximate the concrete semantics of neural networks, which aims at making the reasoning more efficient by using abstract semantics [6]. However, even small precision loss from the abstraction will be enlarged layer by layer so as to bring large precision loss on reachable output set. Thus such a single-pass abstraction based verification is more likely used to verify some shallow (coarse-grained) safety properties. Similar situations may also be encountered when using optimization approaches. A verifier is said to be *complete* if it satisfies that (i) the verifier never returns unknown; and (ii) if the verifier returns violated, the property is actually violate [5]. To perform a complete verification, iterative techniques with tree search are integrated with reachability methods such as [10], and with optimization methods such as [11]–[13]. However, these tree search strategies are of few effective heuristic guidance from the verification process. Henceforth these methods, especially the sound and complete verification methods, still suffer from the lack of efficiency and effectiveness.

In this paper, we present a novel approach to ReLU neural network verification by incrementally determinizing activation states of neurons quickly. The observation behind is that the fewer ReLU neurons with undeterministic activation state (i.e., the neuron may be active or inactive) in hidden layers, the more precise reachable output set will be computed, and henceforth the property is more likely to be verified or falsified. When all ReLU neurons have deterministic states, the precise relations between the input and the output can be computed. Moreover, determinizing these hidden neurons can be specified by constraints over the input variables of the network. Together with a systematic selection of undeterministic neurons from the first hidden layer to determinize, we propose

Manuscript received April 07, 2022; revised June 11, 2022; accepted July 05, 2022. This article was presented at the International Conference on Embedded Software (EMSOFT) 2022 and appeared as part of the ESWEEK-TCAD special issue. This work is supported by the National Natural Science Foundation of China (Nos. 62102432, 62032024, 61872445), and the Natural Science Foundation of Hunan Province of China (No. 2021JJ40697). *(Corresponding authors: Liqian Chen and Ji Wang.)*

B. Yin is with the College of Systems Engineering, National University of Defense Technology, Changsha 410073, China (e-mail:bhyin@nudt.edu.cn).

L. Chen is with the Key Laboratory of Software Engineering for Complex Systems, College of Computer, National University of Defense Technology, Changsha 410073, China (e-mail:lqchen@nudt.edu.cn).

J. Wang is with the State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, Changsha 410073, China (e-mail: wj@nudt.edu.cn).

J. Liu is with the Ant Group, Hangzhou 310000, China (e-mail:jiangchao.ljc@antgroup.com).

a layer-wised input splitting strategy and constraint-guided input refinement to achieve a sound and complete verification approach.

Our main contributions are as follows:

(1) We present a sound and complete verification framework by incrementally determinizing activation states of neurons via layer-wised input splitting and constraint-guided input refinement. This framework is proposed on top of a new verification problem representation combining the benefit of both interval domain (i.e., fast) and relation domain (i.e., precise). Our splitting strategy (splitting guided by neurons in first undeterministic layer) guarantees that each split will determine at least one further undeterministic neurons in all the sub-problems. As far as we know, many methods based on input splitting (dichotomy input splitting [10], gradient-based input splitting [14]) do not have such a guarantee, whereas such a guarantee is important for obtaining verification completeness within a predictable worst-case splitting depth. We are also committed to proposing improvement on linear relaxation (which are usually involved in neural network verification), lightweight backward analysis, speculative constrain-guided input refinement and counter-example generation, which all fit to our framework well and contribute to improve our verifying efficiency.

(2) We implement a parallel neural network verification tool called LayerSAR and evaluate it. Compared to existing state-of-the-art complete verifiers, LayerSAR verifies more properties with less time consumption overall, and achieves at least 114X, 1163X, 846X, 81X, 2.8X and 2.1X speedups respectively on average over Neurify, Planet, Marabou, Venus, Venus2 and nnenum when verifying ACAS Xu networks. Especially, our method are much more effective than others considering generating counter-examples for false properties. The ablation study has also shown the performance improvement benefited from our proposed techniques, that may devote to the community of neural network verification.

II. Preliminaries

In this work, focus our attention on fully-connected, feed-forward neural networks with ReLU activation functions. A neural network $\mathcal{N}$ computes a function $\mathcal{F}_{\mathcal{N}} : \mathbb{R}^n \rightarrow \mathbb{R}^o$, where n is the number of input dimensions and o is the number of output dimensions. The input vector is denoted as $X = \{x_1, x_2, ..., x_n\}$. For each variable x_i, we use $[\underline{x_i}, \overline{x_i}]$ to denote its interval range, where $\underline{x_i}, \overline{x_i}$ are constant values.

1) Abstraction: An abstraction is a computer-representable over-approximation of a possibly unbounded set of values [15]. As in [10], for a neuron s in the hidden layers and the output layer, we use a symbolic interval $[s_l, s_u]$ to abstract its reachable values, where symbolic variables s_l and s_u are called the *symbolic bounds* of s, whose value are linear expressions over input variables. Namely, s_l, s_u are in the form of $\sum_{i=1}^{n} w_i x_i + b$ where $w_i, b \in R$ are constant coefficients, and $x_i \in X$. We also use $[\underline{s_l}, \overline{s_l}]$ to denote the value range of symbolic bound s_l, where $\underline{s_l}, \overline{s_l}$ are constant values which can be computed based on the value bounds of input variables. For instance, if $\underline{s_l} = \sum_{i=1}^{n} w_i x_i + b$, then $\underline{s_l}$ can be computed via $\sum_{i=0}^{n} w_i \overline{x_i^{\pm}} + b$, where $x_i^{\pm} = \overline{x_i}$ if $w_i < 0$, and $x_i^{\pm} = \underline{x_i}$ otherwise. The computation of $\overline{s_l}$ follows the same principle. Similarly, we use $[\underline{s_u}, \overline{s_u}]$ to denote the value range of symbolic bound s_u.

Moreover, for a hidden neuron s with an activation function, we use symbolic variables s^b, s^f to denote its values before and after applying the activation function. And we use two symbolic intervals $[s_l^b, s_u^b]$ and $[s_l^f, s_u^f]$ as the abstraction of neuron s before and after its activation functions respectively.

2) Linear Propagation: Neural networks utilize linear transformations to propagate values through layers. For any neuron s in the first hidden layer, we have

$$s_l^b = s_u^b = \Sigma_{i=1}^{n} w_i x_i + b$$

where $w_1, w_2, ..., w_n$ are the weights of the corresponding edges and b is the bias on this neuron. In general, for a neuron s in the j-th hidden layers or the output layer, we have

$$s_l^b = \sum_{i=1}^{m} w_i y_i^{f\pm} + b$$
$$s_u^b = \sum_{i=1}^{m} w_i y_i^{f\pm'} + b$$

where y_i denote the i-th neuron from the previous layer (i.e., the $(j-1)$-th layer), and

$$y_i^{f\pm} = \begin{cases} (y_i)_u^f & \text{if } w_i < 0 \\ (y_i)_l^f & \text{otherwise} \end{cases} \quad y_i^{f\pm'} = \begin{cases} (y_i)_l^f & \text{if } w_i < 0 \\ (y_i)_u^f & \text{otherwise} \end{cases}$$

3) Linear Relaxation: Liner relaxation is a widely used abstraction for non-linear activation functions [13], [14], [16], [17]. For a ReLU neuron s, since s^f =ReLU$(s^b) = max(s^b, 0)$ is non-linear, we need to take approximation to compute s^f when the sign of s^b is not deterministic. Let $[c, d]$ be the interval value range of s^b, where c, d are constant values. One well-known approach is to use the following two linear constraints instead [14]:

$$\frac{d * s^b}{d - c} \leq s^f, s^f \leq \frac{d * (s^b - c)}{d - c}$$

4) Dependency Analysis: Dependency analysis [18], [19]is a method to further reduce the ReLU space during the"branch and bound" based verification. In dependency analysis, a neuron node s *depends* on another node t if whenever (i.e., for any network input in the given input space) the activation state of t is *deterministic* (i.e., active or inactive), the activation state of s has to be *deterministic* (i.e., active or inactive).

III. Our Approach

A. Our Verification Framework

In this work, we firstly define the *verification problem for neural networks* more clearly as follows.

Definition 1: (Verification Problem for Neural Networks) Given a neural network $\mathcal{N}$, an input range $\mathcal{X} \subseteq \mathbb{R}^n$ of the network, a set of linear constraints on input variables Ω (which equals to a predicate that intersecting all the elements inside), and an property ψ (which is a set of constraints on output variables, then the unsafe set $US = \{y \mid y \in \mathbb{R}^o$ and $y \models \neg\psi\}$), the *verification problem for neural networks* is to check if $Output(\mathcal{N}, \mathcal{X}, \Omega) \cap US = \emptyset$, where $Output(\mathcal{N}, \mathcal{X}, \Omega) = \{y \mid y = \mathcal{F}_{\mathcal{N}}(x), x \in \mathcal{X} \text{ and } x \models \Omega\}$.

Note that, the vector of activation states of hidden neurons, records as $\mathcal{A}$, can be implied from $\mathcal{N}$, $\mathcal{X}$ and Ω. For the sake of improving efficiency of propagation and choosing splitting node, we store it explicitly. Thus we actually record the *verification problem representation* as $\langle \mathcal{N}, \psi, \mathcal{X}, \Omega, \mathcal{A} \rangle$.

The main framework of our approach, named VISIR, is shown in Algorithm 1, which fits to the framework of "bound and branch (B&B)" [20]. VISIR takes as input a neural network verification problem $\langle \mathcal{N}, \psi, \mathcal{X}, \Omega, \mathcal{A} \rangle$ and the *splitting depth sd*. Ω stores a set of linear constraints that are used to constrain the input, i.e., linear input constraint set. For the first time of applying VISIR over a neuron network, Ω is initialized as the input range $\mathcal{X}$ when it is not given. $\mathcal{A}_s$ records the activation state of neuron s. It may take three values: *active*, *inactive* and *unknown*. Each hidden neuron is initialized as $unknown$. We define that s is $deterministic$ if $\mathcal{A}_s$ is *active* or *inactive*, otherwise s is $undeterministic$. sd denotes the depth of input splitting, initialized by 0. The algorithm first calls a single-pass verification (detailed in Sect. III-B) to check whether ψ can be verified (Line 1), which returns a tuple of three elements, i.e., res indicates the verification result (i.e., $True$,$False$ or $unknown$), $\mathcal{R}$ is the reachable set of values of each neuron and Ω' is the updated linear input constraint set. If res is $False$, the algorithm terminates immediately with a *counter-example* violating ψ; if res is $True$, it returns with $True$; if res is $unknown$, our algorithm performs splitting by FUL strategy (Line 8, detailed in Sect. III-C) and speculative input refinement (Line 9, detailed in Sect. III-D) to prepare two sub-problems (i.e., $\langle \mathcal{N}, \psi, \mathcal{X}_1, \Omega_1, \mathcal{A}^1 \rangle$ and $\langle \mathcal{N}, \psi, \mathcal{X}_2, \Omega_2, \mathcal{A}^2 \rangle$) with smaller input regions, and verifies them respectively (Lines 10-11). If both sub-problems are verified, it return $True$ (Line 12), which indicates that the original input verification problem is successfully verified.

Algorithm 1 VISIR (Verification via Iterative Splitting and Input Refinement)

```
Input: Network $\mathcal{N}$, Property $\psi$, Input range $\mathcal{X}$,Linear input constraint $\Omega$, Activation state $\mathcal{A}$, Splitting depth $sd$
Output: $True$ or $False$
 1: ($res$,$\mathcal{R}$,$\Omega'$,$\mathcal{A}'$) ← single_pass_verification($\mathcal{N}, \psi, \mathcal{X}, \Omega, \mathcal{A}$)
 2: if $res = False$ then
 3:     Terminate with $False$ and a counter-example
 4: else if $res = True$ then
 5:     return $True$
 6: else                                              ▷ $res = unknown$
 7:     $sd \leftarrow sd + 1$
 8:     ($\Omega_1$,$\Omega_2$,$\mathcal{A}^1$,$\mathcal{A}^2$) ← splitting_by_FUL($\mathcal{N}$,$\Omega'$,$\mathcal{R}$,$\mathcal{A}'$)
 9:     ($\mathcal{X}_1$, $\mathcal{X}_2$) ← input_refinement($\mathcal{X}$,$\Omega_1$,$\Omega_2$,$sd$)
10:     VISIR($\mathcal{N}, \psi, \mathcal{X}_1, \Omega_1, \mathcal{A}^1, sd$) //sub-problem 1
11:     VISIR($\mathcal{N}, \psi, \mathcal{X}_2, \Omega_2, \mathcal{A}^2, sd$) //sub-problem 2
12:     return $True$
13: end if
```

Our framework is possible to perform (single-pass) abstract analysis only on $\mathcal{X}$ (i.e., interval domain, devoting to a fast "bound" procedure), while to perform splitting and property checking on Ω (i.e., relation domain, devoting to an effective and precise "branch" procedure), which both make our verification framework be efficient.

An Illustrating Example: Now we use the example shown in Figure 1 to illustrate our approach. The network has two input neurons (denoted by symbolic variables x and y), two hidden layers (each with one neuron, i.e., s_1 and s_2), and one output layer (with one neuron s_3). For hidden layers, we show explicitly the symbolic variables representing the values before and after activation. For simplicity, no activation function is applied in the output layer. The weights (resp., biases) are labeled on the edges (resp., under the neurons).

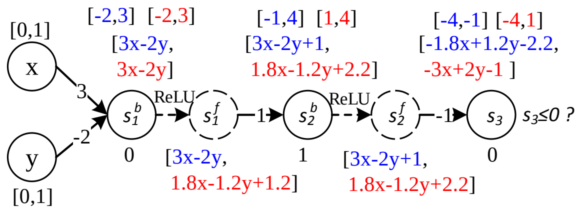

Fig. 1. An illustrating example

As shown in Figure 1, the input range is [0, 1] × [0, 1], and the property to be verified ψ is $s_3 \leq 0$. Our approach first initializes $\mathcal{A}_{s_1} = unknown$, $\mathcal{A}_{s_2} = unknown$ and $\Omega = \{x \geq 0,\ x \leq 1,\ y \geq 0,\ y \leq 1\}$. Then it utilizes abstract analysis (based on abstract interpretation [15]) to compute the reachable set $\mathcal{R}$. The analysis results are annotated above or below each symbolic variable in Figure 1, where the lower (resp., upper) symbolic bounds and their value bounds are shown in blue (resp., red). To verify the property, it first checks the value bounds of s_3, that is $[-4, 1]$, which means the property ψ cannot be verified or falsified yet. Then our approach makes use of the symbolic upper bound of s_3 (i.e., $-3x + 2y - 1$) and checks whether the constraint set $\Omega \cup \{-3x + 2y - 1 > 0\}$ is satisfiable. Then we use a constraint solver (e.g., a linear programming solver) to find an input that satisfies the constraint set. Actually, for this example, after concrete execution, it will turn out that the found input point is not a real counter-example. Thus this single-pass verification process returns *unknown*. Then Ω is updated as $\Omega \cup \{-3x + 2y - 1 > 0\}$.

Then we choose a target neuron in the first undeterministic layer, that is s_1 in this example. By making use of the symbolic bounds over s_1^b, our approach conducts splitting by adding constraints: $3x - 2y \leq 0$ or $3x - 2y > 0$. The splitting generates two sub-problems with the constraint sets $\{\Omega_1 = \Omega \cup \{3x - 2y > 0\},\ \mathcal{A}_{s_1}^1 = active\}$ and $\{\Omega_2 = \Omega \cup \{3x - 2y \leq 0\},\ \mathcal{A}_{s_1}^2 = inactive\}$ respectively. Then the input ranges in both sub-problems are refined, which are guided by the constraints in Ω_1 and Ω_2. After this step, $\mathcal{X}_1 = \emptyset$, since Ω_1 is infeasible, which means that the first sub-problem is verified directly. For the second sub-problem, the input range is updated to a tighter range $\mathcal{X}_2 = [0, 0.34] \times [0.5, 1]$, and at least one more neuron become deterministic (i.e., $\mathcal{A}_{s_1}^1 = active$). Actually, the second sub-problem is then successfully verified by a single-pass verification. Overall, verification problem in

Figure. 1 can be verified by our approach with one time of splitting.

B. Single-pass Verification

1) Abstract Analysis: Abstract analysis propagates an input region through a neural network through abstract semantics (involving linear relaxation, etc.). In this paper, we consider an input region $\mathcal{X}$ as the Cartesian product of n intervals $\mathcal{X} = \mathcal{I}_1 \times \mathcal{I}_2 \times ... \times \mathcal{I}_n$ (where $\mathcal{I}_i = [\underline{x_i}, \overline{x_i}]$, $\underline{x_i}$ and $\overline{x_i}$ are the lower and upper value bounds of the input variable x_i respectively). The abstract analysis takes a neural network and an input region $\mathcal{X}$, then outputs an over-approximation of the reachable set, i.e., *abstract reachable set* $\mathcal{R}$, which formally defined as follows.

Definition 2: (Abstract Reachable Set $\mathcal{R}$) For each neuron s in network $\mathcal{N}$, $\mathcal{R}$ records four intervals, which is defined as $\mathcal{R}_s \triangleq \langle [s_l^b, s_u^b], [s_l^f, s_u^f], [\underline{s_l^b}, \overline{s_l^b}], [\underline{s_u^b}, \overline{s_u^b}] \rangle$, where

- s_l^b and s_u^b (s_l^f and s_u^f) represent the lower and upper symbolic bounds before (after) the activation function respectively, whose value are linear expressions over input variables.
- $\underline{s_l^b}, \overline{s_l^b}$ are the lower and upper value bounds of s_l^b respectively, i.e., $\underline{s_l^b} = min_{\mathcal{X}} s_l^b$ and $\overline{s_l^b} = max_{\mathcal{X}} s_l^b$. Following the same principle, $\underline{s_u^b}, \overline{s_u^b}$ are the lower and upper value bounds of s_u^b.

In addition, our abstraction explicitly records the activation state of each neuron s by $\mathcal{A}_s$. $\mathcal{A}$ records activation states of all hidden neurons in $\mathcal{N}$. And $|\mathcal{A}_D|$ ($|\mathcal{A}_U|$) is defined as the number of *deterministic* (*undeterministic*) neurons. Abstract analysis computes the reachable sets $\mathcal{R}$ from the input layer to output layer layer-wisely. For each hidden neuron s in the network, we first compute $s_l^b, s_u^b, \underline{s_l^b}, \overline{s_l^b}, \underline{s_u^b}$ and $\overline{s_u^b}$, as shown in Sect. 2. Then we update the activation state of s as follows:

$$\mathcal{A}'_s = \begin{cases} inactive & if \left(\overline{s_u^b} \le 0 \text{ } or \text{ } \mathcal{A}_s = inactive\right) \\ active & if \left(\underline{s_l^b} \ge 0 \text{ } or \text{ } \mathcal{A}_s = active\right) \\ unknown & otherwise \end{cases}$$

Note that, some activation states will turn to deterministic from undeterministic here, since this singel-pass analysis has performed on a smaller intput space (due to refined $\mathcal{X}$ and Ω after intput splitting and refinement) than previous singel-pass analysis.

Now we show how to handle activation function ReLU. In the case $\mathcal{A}_s = inactive$ (or $\mathcal{A}_s = active$), no approximation needs to be taken, and then we have $s_l^f = 0, s_u^f = 0$ (or $s_l^f = s_l^b, s_u^f = s_u^b$) respectively. Note that, explicitly recording $\mathcal{A}_s$ can help to improve the efficiency of analysis since we do not need to compute $\mathcal{R}_s$ when $\mathcal{A}_s$ is *active* or *inactive*. When $\mathcal{A}_s$ is *unknown*, since the concrete semantics of ReLU functions is non-linear, linear relaxation need to be taken to compute s_l^f and s_u^f.

Suppose the input symbolic interval is $[s_l^b, s_u^b]$ and the input region is $\mathcal{X}$, the output symbolic interval is $[s_l^f, s_u^f]$. Evaluating which linear relaxation is stronger depends on the comparison metric. There are two widely used metrics: (1) the value range of $[s_l^f, s_u^f]$ satisfying $\mathcal{X}$; (2) the area of $[s_l^f, s_u^f]$ satisfying $\mathcal{X}$. Strictly, which metric is better depends on the property to be verified

Example 1. Suppose the input interval is $[x, 2x]$, where $x \in [-1, 2]$, and the output is y. One linear relaxation outputs symbolic interval $y = [0, (4x + 4)/3]$, whose value range is [0,4] and area is 6, the other outputs symbolic interval $y = [x, (4x + 4)/3]$, whose value range is [-1, 4] and area is 4.5. Considering the property $y \le 0$, it can be found that the metric of value range is better, while considering the property $y \le x$, the metric of area is better.

In our paper, we prefer to use the metric of area for verification. While metric of value range ignores the relations between neurons. It helps reduce the loss of precision during neural network propagation. Therefore, we propose the following linear relaxation for ReLU function:

$$s_u^f = \begin{cases} s_u^b & if \left(\underline{s_u^b} \ge 0\right) \\ \frac{\overline{s_u^b}(s_u^b - \underline{s_u^b})}{\overline{s_u^b} - \underline{s_u^b}} & otherwise \end{cases}$$

$$s_l^f = \begin{cases} s_l^b & if(\text{ } \overline{s_l^b} \ge 0 \land -\underline{s_l^b} < \overline{s_l^b} \text{ }) \\ 0 & otherwise \end{cases}$$

Our linear relaxation can be proved to be an over-approximation of the ReLU function. Note that, under our definition of *abstract reachable set* (i.e., Definition 1), this linear relaxation has achieved smallest area, i.e., it gets smaller area than all the other possible linear relaxations, such as those proposed in DeepPoly [16] and Neurify [14]. In DeepPoly, they do not consider $\overline{s_l^b}$ and $\underline{s_u^b}$, and only make use of the lower value bound and upper value bound of s^b (that is $\underline{s_l^b}$ and $\overline{s_u^b}$ in our case) in linear relaxation. In Neurify, s_l^f is only set as $\frac{\overline{s_u^b} * s^b}{\overline{s_u^b} - \underline{s_l^b}}$ when the activation state is not deterministic.

Example 2. Suppose the input interval is $[x, 2x + 3]$, where $x \in [-1, 2]$. The output symbolic intervals and its area satisfying $x \in [-1, 2]$ are shown in Table I, where column "LR" lists the linear relaxation techniques of ours, DeepPoly and Neurify. It has shown that the symbolic interval area of our linear relaxation technique is the smallest.

TABLE I
OUTPUT COMPARISON OF DIFFERENT LINEAR RELAXATIONS

LR	Output Symbolic Interval	Area
Ours	[x,2x+3]	10.5
DeepPoly	[x,7(x+2)/4]	11.625
Neurify	[2x/3,2x+3]	11

2) Property Checking: After abstract analysis, the reachable sets of the neural network with constrained input is computed (i.e., $\mathcal{R}$ together with Ω). With these information, our algorithm can check whether the property ψ is satisfied. Theoretically, if $(\mathcal{R} \land \Omega) \models \psi$, ψ must be $true$; if $(\mathcal{R} \land \Omega)$ $\models \neg\psi$, ψ must be $false$; otherwise ψ is $unknown$.

For illustration, suppose the property ψ is $y \le y'$, where y and y' are two neurons in the output layer. By our analysis, we can compute the reachable sets of neurons y and y', which consist of $[y_l, y_u]$,$[\underline{y_l}, \overline{y_l}]$,$[\underline{y_u}, \overline{y_u}]$ and $[y'_l, y'_u]$,$[\underline{y'_l}, \overline{y'_l}]$,$[\underline{y'_u}, \overline{y'_u}]$.

Then our algorithm will first check the property with value bounds: If $\overline{y_u} \leq \underline{y'_l}$, then the property must hold; If $\underline{y_l} > \overline{y'_u}$, then the property does not hold, and any input satisfying Ω is a counter-example; In other cases, our algorithm performs further checking by considering symbolic bounds, with the help of constraint solver:

- If $\Omega \cup \{y_u > y'_l\}$ is unsatisfiable, i.e., $y_u \leq y'_l$ always holds, thus the property holds.
- If $\Omega \cup \{y_u > y'_l\}$ is satisfiable, the property does not necessarily hold. In this case, our algorithm finds an input satisfying $\Omega \cup \{y_u > y'_l\}$ by using constraint solver and checks whether it is a counter-example by concrete execution (i.e., by feeding this input to the neural network).
  - If it is a counter-example, the property does not hold.
  - If no counter-example is found, our algorithm conducts further investigation. Since $y_u \leq y'_l$ implies that the property must hold, our algorithm only needs to check the property when $y_u > y'_l$. Thus updated Ω' is $\Omega \cup \{y_u > y'_l\}$. This update mimics a lightweight *backward propagation* from the negation of the property, which can help refine the input ranges. Note that, our algorithm only takes such updation when the negation of the property is linear in our implementation.

Proposition 1: If all hidden neurons are $deterministic$, there is no precision loss during a single pass abstract analysis, and the property can be verified completely. Namely, in Algorithm 1, if $|\mathcal{A}_U| = 0$, res can only be $True$ or $False$ after executing Line 1.

C. Splitting by FUL Strategy

In the following, we first introduce the notion of the *first undeterministic layer*, then describe the splitting process and input refinement.

1) First Undeterministic Layer (FUL): During analysis on a neural network layer-wisely, if all neurons in layer L have deterministic activation states, we call L a *deterministic layer*. Otherwise, L is said to be an *undeterministic layer*. Among the undeterministic layers, we call the first layer with undeterministic neurons as the First Undeterministic Layer (FUL). One property of a neuron s in the FUL is that its symbolic lower bound equals to its symbolic upper bound (i.e., $s_l^b = s_u^b$), which is not necessarily true in later layers. In other word, there is no precision loss for s during the propagation. Of course, the neurons in the layers before FUL also share this property.

2) Splitting with Neurons in FUL : Our strategy is to choose an undeterministic neuron from the FUL as the next splitting target. If there are more than one undeterministic neurons in the FUL, we just choose the neuron with the largest value ranges [1].

[1]The intuition behind is that the larger range of an undeterministic neuron, the larger potential precision loss of propagating it to the later layers. In Sect. IV-C, we have conducted comparison experiments on choosing neurons in FUL with different strategies for splitting.

Algorithm 2 gives the workflow of our FUL neuron based splitting. It takes as input a neural network $\mathcal{N}$, a reachable set $\mathcal{R}$ generated by abstract analysis, a linear constraint set Ω and a vector of activation states of hidden neurons $\mathcal{A}$. In the algorithm, $size(L)$ represents the number of neurons contained in the L-th layer of the neural network $\mathcal{N}$, and suppose all the neurons of the L-th layer are numbered as "$1, 2, ..., size(L)$". N_i^L represents the i-th neuron in the L-th layer. The algorithm is roughly divided into three stages:

1) Based on the activation states $\mathcal{A}$ generated by abstract analysis, determine which layer in the hidden layer is FUL (Lines 1 to 12);
2) Based on the reachable set $\mathcal{R}$ generated by abstract analysis, the neuron with the largest value range is selected from all the neurons in FUL as the target neuron (recorded as t) for generating splitting predicate (Lines 13 to 21);
3) Based on target neuron t, conduct splitting to generate two sub-problems for further verification and return them (Lines 22 to 25). Since the symbolic lower and upper bounds of the target splitting neuron are equal before the activation function (i.e., $t_l^b = t_u^b$), we denote them as t^b for short. Based on t^b, our algorithm generates two constraints $\{t^b \leq 0, t^b > 0\}$. Then the original verification problem is split into two sub-problems $\langle \mathcal{N}, \psi, \mathcal{X}_1, \Omega_1, \mathcal{A}^1 \rangle$ and $\langle \mathcal{N}, \psi, \mathcal{X}_2, \Omega_2, \mathcal{A}^2 \rangle$, where $\Omega_1 = \Omega \cup \{t^b \leq 0\}$ and $\Omega_2 = \Omega \cup \{t^b > 0\}$. And $\mathcal{A}^1$, $\mathcal{A}^2$ are computed as follows: for each neuron s in all hidden layers,

$$\mathcal{A}_s^1 = \begin{cases} inactive & \text{if } s = t \\ \mathcal{A}_s & \text{otherwise} \end{cases}$$

$$\mathcal{A}_s^2 = \begin{cases} active & \text{if } s = t \\ \mathcal{A}_s & \text{otherwise} \end{cases}$$

The activation state of neuron t becomes deterministic after splitting, thus the two sub-problems are easier to be verified by further single-pass verification.

One may consider other strategies to split, e.g., choosing neuron that is not from the FUL to split. Compared with these strategies, our strategy enjoys the following benefits:

- For each splitting, our strategy can always make one undeterministic neuron become deterministic. Less undeterministic neurons will lead to more precise results during abstract analysis. It means that our splitting always makes progress for property verification.
- The predicates generated from the target FUL neuron are usually non-redundant (i.e., if the predicates split Ω into Ω_1 and Ω_2, neither of them is equal to Ω) and mutually-exclusive (i.e., if the predicates split Ω into Ω_1 and Ω_2, $\Omega_1 \wedge \Omega_2 = \perp$). Our strategy can make full use of these characteristics to achieve efficient splitting process, since our verification problem presentation records Ω explicitly.
- With our strategy, the single-pass analysis during the current verification problem can make use of the analysis result of its parent verification problem, e.g., the computation of activation state $\mathcal{A}$, and the computation of abstract

Algorithm 2 FUL neuron based splitting algorithm

```
Input: Neural network $\mathcal{N}$, Reachable Set $\mathcal{R}$,Linear constraint set $\Omega$, Activation state $\mathcal{A}$
Output: $\Omega_1, \Omega_2, \mathcal{A}^1, \mathcal{A}^2$
1: $FUL \leftarrow 1$
2: for $L \leftarrow 1$ to $m$ do
3:     for $s \leftarrow N_1^L$ to $N_{size(L)}^L$ do
4:         if $\mathcal{A}_s = unknown$ then
5:             $FUL \leftarrow L$
6:             break
7:         end if
8:     end for
9:     if $FUL > 1$ then
10:        break
11:    end if
12: end for
13: $max\_interval \leftarrow 0$
14: for $s \leftarrow N_1^{FUL}$ to $N_{size(FUL)}^{FUL}$ do
15:     if $\mathcal{A}_s = unknown$ then
16:         if $\overline{s_u^b} - \underline{s_l^b} > max\_interval$ then
17:             $max\_interval \leftarrow \overline{s_u^b} - \underline{s_l^b}$
18:             $t \leftarrow s$
19:         end if
20:     end if
21: end for
22: $\mathcal{A}^1 \leftarrow \mathcal{A}; \mathcal{A}^2 \leftarrow \mathcal{A}$
23: $\Omega_1 \leftarrow \Omega \cup \{t^b \leq 0\}; \mathcal{A}_t^1 \leftarrow inactive$
24: $\Omega_2 \leftarrow \Omega \cup \{t^b > 0\}; \mathcal{A}_t^2 \leftarrow active$
25: return $\Omega_1, \Omega_2, \mathcal{A}^1, \mathcal{A}^2$
```

reachable set $\mathcal{R}$ for neurons (before the splitting layer), which makes our single-pass analysis more efficient.

- Since the precision loss will be amplified layer by layer during abstract analysis, improving precision of earlier neurons will be more effective in limiting the precision loss in outputs.
- The two sub-problems obtained by FUL neuron based splitting are completely independent, thus they are more conducive for parallel verification to improve the efficiency.

From above, we can get the following proposition directly.

Proposition 2: After one splitting by the FUL strategy, at least one more undeterministic neuron become deterministic when conducting another single pass abstract analysis for each sub-problem.

D. Speculative Constraint-guided Input Refinement

Given the input ranges, abstract analysis computes the reachable sets for each neuron. Since the precision loss will be amplified layer by layer during abstract analysis, the value bounds of the input layer (i.e., earliest layer) can affect the precise of abstract analysis greatly. With tighter input ranges, abstract analysis can find more neurons deterministic and compute more accurate reachable sets, which greatly help to reach high-performance verification (see experimental results of Sect. IV-C4). During each splitting, new constraints are added into Ω, the input ranges can be refined simultaneously. This is achieved by linear programming. More clearly, for each input variable $x_i \in X(i = 0, 1, ..., n)$, our algorithm generates two linear programming problems, where Ω is used as constraints, and $max\ x_i$ and $min\ x_i$ are objective functions. By solving these problems, the input range of x_i is refined.

In the above refinement process, after each splitting, our algorithm needs to call $2n$ times of linear programming solvers to refine all input variables. Since each splitting adds only one constraint (sometimes it may even be redundant), performing input refinement immediately may not lead to significant refinement on the input. Therefore, in order to reduce the overhead of calling the linear program solver, our algorithm utilizes a *speculative input refining* strategy. The main idea lies in that we do not refine the input after each splitting, but every m ($m \geq 1$) times of splitting, where m is configurable. The choice of m is tricky. As m increases, the average running time for each iteration may decrease, but the number of iterations is likely to increase.

Generating Counter-examples from Refined Input: To improve the efficiency of finding counter-examples, after each refinement on the input, our algorithm selects several concrete input points as potential counter-examples for concrete execution. Currently, we choose inputs from the boundaries and middle points, e.g., $(\underline{x_1}, ..., \underline{x_n})$, $(\overline{x_1}, ..., \overline{x_n})$ and $(\frac{\underline{x_1}+\overline{x_1}}{2}, ..., \frac{\underline{x_n}+\overline{x_n}}{2})$. Combining concrete execution and abstraction refinement makes our algorithm more efficient in falsifying the property when the property does not hold.

Since the input ranges of variables are refined each time, we can generating new potential counter-examples from the refined input, which makes out approach be more likely to find true counter-example for false property compared to verification with no input refining.

E. Completeness Discussion

First, our approach is sound (i.e., if the verification process terminates within time-limit and returns "true", then the neural network must satisfy the property), since we use over-approximations of the concrete semantics of neural networks under the framework of abstract interpretation [15], like other similar works utilizing abstractions [6], [16]. In the following, we discuss the completeness.

Theorem 1: Let d be the number of undeterministic neurons of a neural network $\mathcal{N}$ after the first single pass verification. Our approach is complete, in the sense that if $\mathcal{N}$ satisfies the property ψ, then VISIR will terminate within maximal splitting depth d and return "True". In other words, the maximal value of sd in Algorithm 1 is never larger than d.

Proof 1: From Proposition 2, we find that VISIR will make at least one more undeterministic neuron become deterministic in both sub-problems after each splitting and refinement. Namely, if sd is increased by 1, $|\mathcal{A}_U|$ will be decreased by 1 at least after conducting abstract analysis in both sub-problem 1 and sub-problem 2. Thus when sd is increased by d or even larger, $|\mathcal{A}_U|$ will be decreased by d at least, which means all the undeterministic neurons have become deterministic. Then from Proposition 1, property ψ must be verified at that time.

Note that for those approaches not taking the FUL strategy (e.g., [14]) under our framework, there is no guarantee of decreasing the number of undeterministic neurons after each splitting, and thus there is no guarantee of completeness within d splitting steps.

Example 3. We give an example as shown in Figure 2 to show the incompleteness of non-FUL strategy within 2 splitting steps. The network contains one input x, two hidden layers (each with one neuron, i.e., s_1 and s_2 respectively), one output y. The weights are $s_1 = x$, $s_2 = 4s_1 - 1$, $y = s_2$. Assume $x \in [-1, 1]$ and property ψ is $y >= (4x - 1)/2$. Ω is initialized as $\{x \geq -1,\ x \leq 1\}$

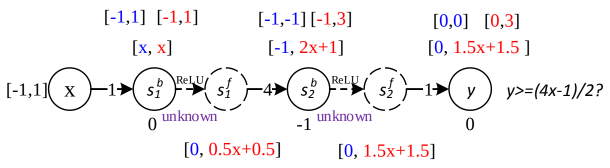

Fig. 2. Example for completeness discussion

After the first time of abstract analysis, we get $s_1^b = [x, x], s_1^f = [0, (x+1)/2], s_2^b = [-1, 2x+1], s_2^f = [0, 3(x+1)/2], y = [0, 3(x+1)/2]$ as shown in Figure 2. Since whether ψ holds is unknown for $x = [-1, 1]$, we perform splitting. Suppose we split with the (non-FUL) neuron s_2 by its bound s_2^b, and get two cases: (1) $\Omega_1 = \Omega \cup \{2x + 1 < 0\}$; (2) $\Omega_1 = \Omega \cup \{2x + 1 >= 0\}$. Case (1) is verified by one other single-pass abstract verification directly. For case (2), x is refined to $[-0.5, 1]$, the activation state of neuron s_2 is still not deterministic. With another abstract analysis, we get $s_1^b = [x, x], s_1^f = [x, (x+1)/2], s_2^b = [4x-1, 2x+1], s_2^f = [0, 2x+1], y = [0, 2x+1]$. Whether ψ holds is still unknown. We choose to split over each node at most once, and we continue to split node s_1 and get two cases (21) $\Omega_{21} = \Omega_2 \cup \{x < 0\}$; (22) $\Omega_{22} = \Omega_2 \cup \{x >= 0\}$. Case (21) is easily verified. For case (22), x is refined as $[0, 1]$, but after abstract analysis, we only get: $s_1^b = [x, x]$, $s_1^f = [x, x], s_2^b = [4x-1, 4x-1], s_2^f = [4x-1, 3x], y = [4x-1, 3x]$. The activation state of s_2 and whether ψ holds are still unknown, which means the property can not be verified by non-FUL strategy with at most one split on each node, i.e., within 2 splitting steps.

IV. Evaluation

To evaluate our approach, we have implemented a prototype LayerSAR on top of Neurify [14]. In the implementation, we use the OpenBLAS library to efficiently calculate matrix multiplication. During the phases of single-pass verification and input refining, we use lp_solve 5.5 [21] to check whether a constraint set is satisfiable and solve optimization problems. In each splitting, the two new sub-problems can be verified independently, thus they are verified with multi-threading to gain speedup.

A. Experimental Setup

In the experiments, we use following datasets: ACAS Xu [22], Collision Detection [13] and MNIST [23]. The dimensions of all of the verification problems from above four datasets are given in Table II.

- The ACAS Xu data set is a neural network based advisory system for aircraft in order to prevent collisions.It consists of 45 networks, each with 4 properties (Properties 1 to 4 defined in [11]) to be verified.
- The Collision Detection (row "CD") network is used to predict collisions between two vehicles with different configurations. It contains 500 verification problems. For MaxPooling activation function, we decompose it into a series of ReLUs as shown in [20].
- We also conduct preliminary experiments on MNIST dataset, which is designed for classifying hand-written digits. It has more inputs than the above datasets.

TABLE II
Dimensions of our benchmarks

		Model Architecture		
Dataset	#Properties	#Inputs	#Hidden units	#Outputs
ACAS Xu	180	5	6 layers*50	5
CD	500	6	40 MaxPooling+19 ReLU	2
MNIST	700	784	2 layers * 512	10

All experiments are carried out on a computer with 16GB RAM, a 3.6 GHz octa-core Intel® Core™ i7-7700U host CPU, and Ubuntu 16.04. The number of maximum threads is set to 128 for LayerSAR.

B. Comparisons with Other Complete Verifiers

1) Experiments on Safety: For the comparisons we restrict our attention to complete verifiers. These verifiers are often less scalable than incomplete ones, while they provide full guarantees on the correctness of their outputs, which is a key objective in safety-critical areas. We conduct comparison experiments with some typical, technically similar and high-performance tools, e.g., Neurify [14], Planet [13], Marabou [12], Venus [18], Venus2 [19], nnenum [24], [25] and complete version of ERAN with DeepPoly domain [16]. Neurify is one of the most similar tool with ours, which combines symbolic linear relaxation and gradient guided input splitting. Planet and Marabou both perform complete verification of neural networks with the help of SMT. Marabou is built based on Reluplex [11], also performing input splitting heuristically to achieve high efficiency. Venus2, which is on top of Venus, is a MILP-based verifier combining input splitting and symbolic interval propagation, leveraging dependency relations between the ReLU nodes to reduce the search space. Nnenum combines zonotopes with star set overapproximations, and uses efficient parallelized ReLU case splitting.

In our experiment, Planet (not support multi-threading) and Marabou were run with the parameters reported in [12] (using Marabou's D&C model with 64 cores on its stable branch, i.e., cav_artifact[2]). Neurify and ERAN was run with MAX_THREAD set to 128 and Venus was run with 7 splitters,

[2]https://github.com/NeuralNetworkVerification/Marabou/tree/cav_artifact

TABLE III
Comparison with complete verifiers

Tools \ Dataset		ACAS Xu P1	P2	P3	P4	Total	CD
LayerSAR	F/T/TO	0/45/0	39/6/0	3/42/0	3/42/0	45/135/0	172/328/0
	Time(s)	108	63	11	6	188	12
Neurify	F/T/TO	0/45/0	36/4/5	3/42/0	3/42/0	42/133/5	-
	Time(s)	572	19821	997	53	21443	-
Planet	F/T/TO	0/24/21	20/0/25	3/42/0	3/40/2	26/106/48	172/328/0
	Time(s)	96075	98041	13505	11084	218705	96
Marabou	F/T/TO	0/29/16	38/2/5	3/39/3	3/40/2	44/110/26	172/328/0
	Time(s)	87496	31120	27461	13046	159123	241
Venus	F/T/TO	0/45/0	39/6/0	3/42/0	3/42/0	45/135/0	-
	Time(s)	8362	2698	290	3839	15189	-
Venus2	F/T/TO	0/45/0	39/6/0	3/42/0	3/42/0	45/135/0	-
	Time(s)	111	389	15	12	527	-
nnenum	F/T/TO	0/45/0	39/6/0	3/42/0	3/42/0	45/135/0	-
	Time(s)	216	109	40	35	400	-
LayerSAR Speed-up	vs.Neurify	5.3	314.6	90.6	8.8	114.1	-
	vs.Planet	889.6	1556.2	1227.7	1847.3	1163.3	8
	vs.Marabou	810.1	494	2496.5	2174.3	846.4	20.1
	vs.Venus	77.4	42.8	26.4	639.8	80.8	-
	vs.Venus2	1.0	6.2	1.4	2.0	2.8	-
	vs.nnenum	2.0	1.7	3.6	5.8	2.1	-

128 workers (i.e., 128 threads). Venus2 was run with its version of participating in the VNN-COMP 2021 [26][3]. Nnenum was run with the online version with default parameters directly[4]. Neurify, Venus, Venus2, nnenum and ERAN were not run on the CollisionDetection, due to lack of frontend support of the dataset. All the experiments were conducted with a time limit of 3600s.

The experimental results are shown in Table III. These tools run on two date sets: ACAS Xu (columns "P1, P2, P3, P4, Total"), Collision Detection (row "CD"). Row "F/T/TO" means the number of verified unsatisfiable (i.e., false) properties, satisfiable (i.e., true) properties and unverified properties (i.e., timeout) respectively. The row "Time" includes the total running time on all networks. In the case of a timeout, the runtime is counted as the time limit (3600s), even though the real runtime could be more. As a result, the total runtime for methods with more timeout cases would be worse in practice. Since our approach is with the least unverified properties, the speedups compared to other tools in practice could be larger than these reported here.

For ACAS Xu networks, LayerSAR outperforms other tools by verifying more problems and by being faster on average. LayerSAR, Venus and nnenum verify all of the 180 verification problems, while Marabou, Planet and Neurify have timeout cases. LayerSAR at least has an average 114X, 1163X, 846X and 81X speedups over Neurify, Planet, Marabou and Venus respectively. To evaluate the efficiency of counter-examples generation for false property, we further summarize the total time cost for the 45 unsatisfiable properties, which is 19s for LayerSAR, 12540s for Neurify, 76516s for Planet, 12083s for Marabou and 950s for Venus. Thus LayerSAR at least achieves 660X, 4026X, 635X and 50X speedups over them respectively. This achievement mainly comes from our counter-example generating method of taking concrete testing on refined input. After several splittings, our approach can narrow the search space to a quite small input space, which is more conducive to finding counter-examples, compared to many method of doing testing directly on the initial input space. Compared with nnenum (the fastest complete verifier on ACAS Xu networks in VNN-COMP 2021 [26]) and Venus2, LayerSAR has the same order of magnitude on verification efficiency as them, and still achieves an average 2.8X and 2.1X speedups for all the 180 properties (17.6X and 2.4X speedups for the 45 unsatisfiable properties) respectively.

We have also conducted a comparison between (the complete version of) ERAN with DeepPoly domain and our LayerSAR on ACAS Xu dataset with property 1-4. LayerSAR verified 180 properties (45 false and 135 true) in 183s, while ERAN verified 130 properties in 754.3s.[5]

We further compared these verifiers in Figure 3, which summarizes the execution of all 180 properties from ACAS Xu. Here, the y-axis is the time in seconds (with a time limit 3600s), and the x-axis is the number of properties verified within that time. Notice that the y-axis is log scale, so that differences in runtimes between easy and hard benchmark instances are both visible. The result has shown more clearly that LayerSAR outperforms other verifiers on ACAS Xu.

For Collision Detection networks, it has shown that all the three verifies finished successfully. On this easily verified data set, LayerSAR still obtains 8X and 20.1X speedups over Planet and Marabou on average respectively.

[3]https://github.com/pkouvaros/venus2_vnncomp21

[4]https://github.com/stanleybak/nnenum. We run on a recently download of the master branch.

[5]For 5 true and 45 false properties, ERAN terminates with exception errors quickly (thus we did not put the result in Table III).

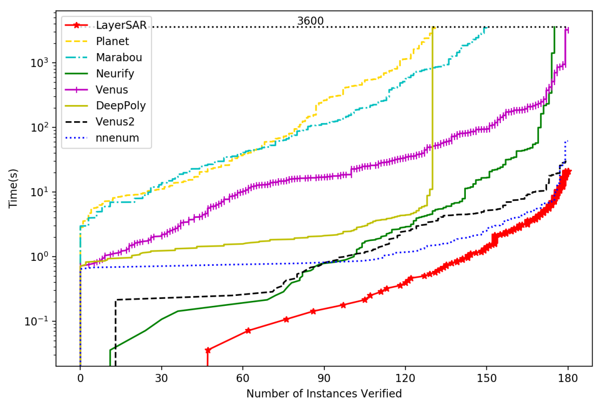

Fig. 3. Comparison of Complete Verifiers on ACAS Xu

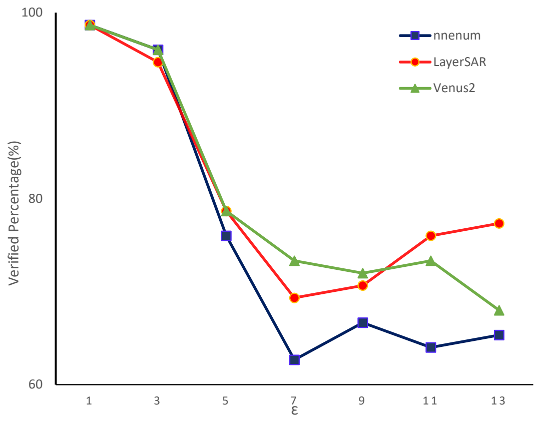

Fig. 4. The verifying percentage with different ε on MNIST

2) Experiments on Robustness: We used MNIST [23] for our preliminary experiments. MNIST consists of grayscale images of size 28 $\times$ 28 pixels (i.e., 784 inputs) with 10 likely output. We conduct experiments with an experimental setup similar to VNN-COMP2021 [26]. The benchmark set consists of three fully-connected MNIST networks with 2 layers and 24, 50 and 512 ReLU nodes in each layer. We randomly sampled 25 correctly classified images from the MNIST test set. For each image, we perform perturbation to every pixel value with a total perturbation bound ε (its range is 1 to 13) by the L_∞ norm (i.e., $|X|_\infty < \varepsilon$) [10]. The timeout bound is 600s for each verification. We conduct experiments on the top 3 tools in Table III and Figure 3, i.e., LayerSAR, Venus2 and nnenum [6].

Figure 4 shows the results. When ε is small (i.e., less than 3), all the 3 tools can verify most of the safe properties. But as ε get larger (i.e., less than 7), the number of verified safe cases drops dramatically because (1) the underlying model tends to become unsafe and (2) LayerSAR suffers from relatively higher overestimation errors, since our singe-pass analysis with symbolic interval is less precise than that of nnenum (with star set) and Venus (with MILP). However, as ε increases further, more and more counter-examples can be found by LayerSAR. This result indicates that LayerSAR can generate counter-examples more efficiently than nnenum and Venus2, by taking our FUL splitting and input refinement.

C. Ablation Study

To evaluate the contribution of each technique, we conduct the following experiments on verifying 4 properties of ACAS Xu networks. All the experiments were conducted without time limit.

1) Linear Relaxation: We compare our linear relaxation technique with those from DeepPoly [16] and Neurify [14]. We have implemented all these linear relaxation techniques in our framework to conduct a fair comparison. The results are listed in Table IV. It demonstrate that our linear relaxation is more efficient than the other two in verifying 4 properties of ACAS Xu networks. The average speedup is around 1.4 times. This achievement is mainly obtained from that the output symbolic interval of our linear relaxation technique has introduced smallest area within the input region.

2) Splitting with Neuron by FUL: To evaluate our splitting strategy with neurons in the FUL, we compare our strategy (column "FUL") with three non-FUL strategies: a total random strategy (column "RD", which randomly chooses one neuron for splitting from all undeterministic neurons), the gradient based branching [14], [27] (column "GD", which computes scores based on gradient information to prioritize undeterministic ReLU nodes) and the dependency analysis based branching [18], [19] (column "DD"). DD branching strategy is based on intra-layer dependencies (since inter-layer dependencies analysis is too costly [18]), i.e., DD firstly specifies a non-FUL splitting layer, then chooses the neuron with largest *depending number* in the selected layer, where *depending number* of neuron s is defined as the number of neurons that depend on s in its same layer. All the branching strategies are implemented in our tool to assure the fairness. Note that, for the non-FUL strategies, we split over each node at most once for each property.

The result is shown in Table V. When verifying the property P2, there are 17, 21 and 20 networks that cannot be verified by the random, gradient based and dependency based strategies respectively, because the non-FUL strategies based verification is incomplete within our assumption. Considering the time cost, the results show that FUL is much faster than the non-FUL strategies, with over 1-2 orders of magnitude speedup on average for these networks.

3) Strategies for choosing splitting neuron in FUL: We compare our strategy of choosing neuron with the largest value range (column "LVR") in FUL with following strategies: (1) choosing the first undeterministic neuron (i.e., with the smallest index, column "FUN"); (2) choosing neuron with the smallest value range (column "SVR"); (3) choosing the neuron with largest *depending number* (column "LDN") in FUL.

The results are listed in Table VI. It demonstrate that

[6]Note that, LayerSAR supports "nnet" format, while nnenum and Venus2 support "onnx" format, thus we have translated the MNIST models from "nnet" format into "onnx" format with tool NNet, https://github.com/sisl/nnet.

TABLE IV
COMPARISON OF DIFFERENT LINEAR RELAXATIONS

	Total Time			Our LR Speed-up	
ACAS Xu	Our LR	DeepPoly's LR	Neurify'LR	vs. DeepPoly'LR	vs. Neurify'LR
P1	108	170	153	1.57	1.42
P2	63	80	85	1.27	1.35
P3	11	13	15	1.18	1.36
P4	6	7	7	1.17	1.17
Total	188	270	260	1.44	1.38

TABLE V
COMPARISON OF FUL AND NON-FUL SPLITTING STRATEGIES

	U/S/T				Total Time				FUL's Speed-up		
ACAS Xu	FUL	RD	GD	DD	FUL	RD	GD	DD	vs. RD	vs. GD	vs. DD
P1	0/45/0	0/45/0	0/45/0	0/45/0	108	25971	1642	1332	240	15	12
P2	39/6/0	28/0/17	24/0/21	25/0/20	63	46167	5305	3415	733	84	54
P3	3/42/0	3/42/0	3/42/0	3/42/0	11	814	68	135	74	6	12
P4	3/42/0	3/42/0	3/42/0	3/42/0	6	289	51	96	48	9	16
Total	45/135/0	34/129/17	30/129/21	31/129/20	188	73241	7066	4978	390	38	26

TABLE VI
COMPARISON OF STRATEGIES FOR CHOOSING SPLITTING NEURON IN FUL

ACAS Xu	Total Verifying Time				LVR's Speed-up		
	LVR	FUN	SVR	LDN	vs. FUN	vs. SVR	vs. LDN
P1	108	139	159	117	1.3	1.5	1.1
P2	63	75	77	78	1.2	1.2	1.2
P3	11	14	17	13	1.3	1.5	1.2
P4	6	7	8	7	1.2	1.3	1.2
Total	188	235	261	215	1.3	1.4	1.1

choosing neuron with the largest value range (column "LVR") in FUL is a little more efficient than the other strategies on average when verifying the 4 properties of ACAS Xu, even better than the intra-layer dependency based strategy (column "LDN") [18]. It indicates that the value ranges of neurons may affect much on the verification efficiency, since its precision loss will be amplified layer by layer during abstract analysis.

4) Strategies for Input Refinement: We first compare our default input refinement strategy (refining all the input after each splitting, called *AIR*) with a strategy called *PIR*, which only refines partial of the input (i.e., the first variable) after splitting[7]. The comparison result is shown in Table VII. Both strategies can verify all the networks in ACAS Xu. The total verifying time of AIR is significantly decreased compared with that of PIR strategy. The rate of decrease in iterations and the speedup in verification time are both nearly 3 orders of magnitude on average.

We analyze the influences of speculative number m of input splitting when verifying ACAS Xu. In Figure 5, the x-axis shows the different speculative number m and the y-axis shows the speedup of our default strategy (i.e., $m = 1$) relative to experiments with different values of m. The results show that, for property 1 and 2, $m = 1$ achieves the highest efficiency, while for property 3 and 4, $m = 2$ achieves the highest efficiency. This indicates that LayerSAR can be more efficient with a suitable speculative number, and the choice of m may be sensitive to the structure of neural network and property.

[7]Note that, if we don't do input refinement for any variables, it failed to verify most of the properties.

TABLE VII
COMPARISON OF DIFFERENT INPUT REFINEMENT STRATEGIES

	Number of Iterations		Total Time		
ACAS Xu	AIR	PIR	AIR	PIR	Speedup
P1	720784	1236688516	108	368586	3413
P2	374527	236213960	63	66806	1060
P3	50647	4805628	11	777	71
P4	17752	1014266	6	161	27
Total	1163710	1478722370	188	436330	2321

5) Multi-Threading Technique: For ACAS Xu date set, we set the thread number (i.e.,"#Threads") as 1, 4, 16, 64, 128 separately, and record the verifying results in Table VIII, which indicates that our approach is naturally parallelized.

TABLE VIII
THE IMPACT OF THREAD NUMBERS

#Threads	1	4	16	64	128
Total time(s)	473	383	262	205	188

D. Threats to Validity

The main threat to validity lies in the representativeness of the benchmarks. The datasets used in this paper have relatively

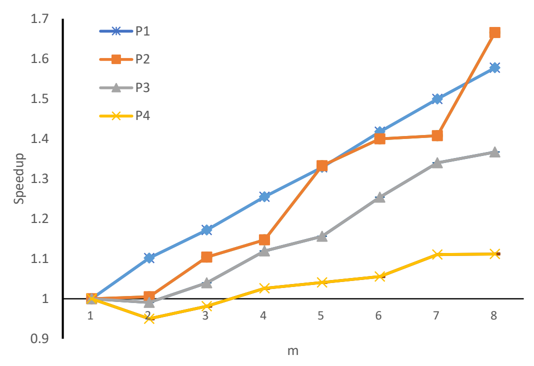

Fig. 5. The speedups with different speculative numbers

small number of neurons (with ReLU activation function). Because of that, our findings may be less convincing. However, existing *complete* verifiers are known to be hard in verifying large-scale neural networks at moment, and the size of neural networks in the evaluation dataset used in this paper is comparable with those used in existing state-of-the-art complete verification tools. Thus comparing with complete verifiers over these data sets still show meaningful results.

V. Related Work

Huang et al. [9] and Liu et al. [5] survey techniques to verify safety and robustness of deep neural networks. Besides the soundness as the basic requirements of verification, all these methods can be either incomplete or complete.

It is generally impractical to verify the property of the neural network through exact reasoning directly. To reduce the complexity of verification, a set of abstraction methods are applied during the neural network propagation, such as symbolic interval analysis [10], abstract interpretation [6], [16] and duality [28], star sets based reachability analysis [29]. Convex relaxation is another widely used technique for abstraction [13], [14], [16], [17], [30]. Botoeva et al. [31] presented a layer-wise convex relaxation framework that unifies all LP-relaxed verifiers and suggested there is an inherent barrier to tight verification for existing convex relaxations. Most of these researches are often incomplete, and may scalable for large networks. While unlike complete methods, incomplete method may not give conclusive answers for verification problems due to precision loss during abstraction.

In this paper, we prefer the complete approaches. Such approaches are the first batch of verification methods for neural networks. The ideas are to encode the verification problems and feed them to Satisfiability Modulo Theory [7], [11]–[13] or Mixed Integer Linear Programming (MILP) solvers [32], [33]. Recently, some researchers use a combination of overestimation and refinement techniques to get a complete verification [10], [12], [14], [18], [34]–[41]. Bunel et al. [20] present a unified "branch and bound" view of piecewise linear neural network verification, and well fits SMT-based verifiers (e.g., Planet, ReluPlex) to their framework.

Splitting over input space or acivation neuron is often used for tightening the output of neural networks [10], [12], [16], [27], [37], [42]–[46]. Two of the most closely related works are ReluVal [10] and Neurify [14]. To verify a given property, both of them have tried over-approximated technique to compute reachable output sets and applied iteration techniques to refine the outputs. ReluVal [10] performs symbolic interval analysis and splits input space by dichotomy strategy. Neurify [14] combines symbolic linear relaxation and intermediate predicates based splitting techniques, where splitting neurons are ordered statically by gradient analysis before verification. The key differences to our approach are that we always choose FUL neuron to generate splitting predicates dynamically, and utilize constraint guided input refinement technique to make more undeterministic neurons become deterministic. That makes our search tree has smaller maximum depth and obtain a much better performance.

Several researchers have devoted to utilize abstract interpretation [15], [16], [47], [48], efficient bound propagation techniques [40], [41], [49] and scalable convex hull approximations [50] for getting precise reachable output efficiently, These works aim to generate more useful output for abstract analysis from different aspects. They are orthogonal to our work, since we use single-pass abstract analysis as a black box. It is interesting to combine these techniques with our work in the future.

VI. Conclusion

We have presented an approach for verifying safety and robustness properties of neural networks through determinizing activation states of neurons. During the verification, we propose a more precise linear relaxation technique, a FUL neuron based splitting strategy and a configurable constraint-guided input refinement to make more neurons with undeterministic activation states become determinized, which all contribute to achieve high verifying efficiency of our approach. And our approach can achieve complete verification finally.

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[2] C. Bloom, J. Tan, J. Ramjohn, and L. Bauer, "Self-driving cars and data collection: Privacy perceptions of networked autonomous vehicles," in *Thirteenth Symposium on Usable Privacy and Security, SOUPS 2017, Santa Clara, CA, USA, July 12-14, 2017*. USENIX Association, 2017, pp. 357–375.

[3] F. Amato, A. López, E. M. Peña-Méndez, P. Vaňhara, A. Hampl, and J. Havel, "Artificial neural networks in medical diagnosis," *Journal of Applied Biomedicine*, 2013.

[4] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos, "Next-generation airborne collision avoidance system," Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, Tech. Rep., 2012.

[5] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, M. J. Kochenderfer *et al.*, "Algorithms for verifying deep neural networks," *Foundations and Trends® in Optimization*, vol. 4, no. 3-4, pp. 244–404, 2021.

[6] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *IEEE Symposium on Security and Privacy (SP 2018)*. IEEE, 2018, pp. 3–18.

[7] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification (CAV 2017)*. Springer, 2017, pp. 3–29.

[8] W. Xiang, H.-D. Tran, and T. T. Johnson, “Output reachable set estimation and verification for multilayer neural networks,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5777–5783, 2018.

[9] X. Huang, D. Kroening, M. Kwiatkowska, W. Ruan, Y. Sun, E. Thamo, M. Wu, and X. Yi, “Safety and trustworthiness of deep neural networks: A survey,” *arXiv preprint arXiv:1812.08342*, 2018.

[10] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, “Formal security analysis of neural networks using symbolic intervals,” in *27th USENIX Security Symposium (USENIX Security 2018)*, 2018, pp. 1599–1614.

[11] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, “Reluplex: An efficient smt solver for verifying deep neural networks,” in *International Conference on Computer Aided Verification (CAV2017)*. Springer, 2017, pp. 97–117.

[12] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić *et al.*, “The marabou framework for verification and analysis of deep neural networks,” in *International Conference on Computer Aided Verification (CAV 2019)*. Springer, 2019, pp. 443–452.

[13] R. Ehlers, “Formal verification of piece-wise linear feed-forward neural networks,” in *International Symposium on Automated Technology for Verification and Analysis (ATVA 2017)*. Springer, 2017, pp. 269–286.

[14] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, “Efficient formal safety analysis of neural networks,” in *Advances in Neural Information Processing Systems (NIPS 2018)*, 2018, pp. 6367–6377.

[15] P. Cousot and R. Cousot, “Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints,” in *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 1977, pp. 238–252.

[16] G. Singh, T. Gehr, M. Püschel, and M. Vechev, “An abstract domain for certifying neural networks,” *Proceedings of the ACM on Programming Languages*, no. POPL, p. 41, 2019.

[17] W. Lin, Z. Yang, X. Chen, Q. Zhao, X. Li, Z. Liu, and J. He, “Robustness verification of classification deep neural networks via linear programming,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 11 418–11 427.

[18] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener, “Efficient verification of relu-based neural networks via dependency analysis,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3291–3299.

[19] P. Kouvaros and A. Lomuscio, “Towards scalable complete verification of relu neural networks via dependency-based branching,” in *International Joint Conference on Artificial Intelligence (IJCAI21)*, 2021, pp. 2643–2650.

[20] R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. K. Mudigonda, “A unified view of piecewise linear neural network verification,” *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[21] M. Berkelaar, K. Eikland, and P. Notebaert, “lp_solve 5.5, open source (mixed-integer) linear programming system,” 2004.

[22] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, “Policy compression for aircraft collision avoidance systems,” in *IEEE/AIAA 35th Digital Avionics Systems Conference (DASC 2016)*. IEEE, 2016, pp. 1–10.

[23] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] S. Bak, H.-D. Tran, K. Hobbs, and T. T. Johnson, “Improved geometric path enumeration for verifying relu neural networks,” in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 66–96.

[25] S. Bak, “nnenum: Verification of relu neural networks with optimized abstraction refinement,” in *NASA Formal Methods Symposium*. Springer, 2021, pp. 19–36.

[26] S. Bak, C. Liu, and T. Johnson, “The second international verification of neural networks competition (vnn-comp 2021): Summary and results,” *arXiv preprint arXiv:2109.00498*, 2021.

[27] A. Rössig and M. Petkovic, “Advances in verification of relu neural networks,” *Journal of Global Optimization*, vol. 81, no. 1, pp. 109–152, 2021.

[28] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, “A dual approach to scalable verification of deep networks.” in *UAI*, vol. 1, no. 2, 2018, p. 3.

[29] H.-D. Tran, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, “Star-based reachability analysis of deep neural networks,” in *International Symposium on Formal Methods*. Springer, 2019, pp. 670–686.

[30] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel, “Towards fast computation of certified robustness for relu networks,” *arXiv preprint arXiv:1804.09699*, 2018.

[31] H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang, “A convex relaxation barrier to tight robustness verification of neural networks,” in *Advances in Neural Information Processing Systems (NIPS 2019)*, 2019, pp. 9832–9842.

[32] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, “Output range analysis for deep feedforward neural networks,” in *NASA Formal Methods Symposium*. Springer, 2018, pp. 121–138.

[33] V. Tjeng, K. Xiao, and R. Tedrake, “Evaluating robustness of neural networks with mixed integer programming,” *arXiv preprint arXiv:1711.07356*, 2017.

[34] J. Lu and M. P. Kumar, “Neural network branching for neural network verification,” *arXiv preprint arXiv:1912.01329*, 2019.

[35] S. Bak, “Execution-guided overapproximation (ego) for improving scalability of neural network verification,” 2020.

[36] H.-D. Tran, X. Yang, D. Manzanas, P. Musau, L. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, “Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems,” in *Proceedings of the 32nd International Conference on Computer Aided Verification*. Springer, 2020.

[37] G. Anderson, S. Pailoor, I. Dillig, and S. Chaudhuri, “Optimization and abstraction: a synergistic approach for analyzing neural network robustness,” in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 731–744.

[38] Y. Y. Elboher, J. Gottschlich, and G. Katz, “An abstraction-based framework for neural network verification,” in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 43–65.

[39] H.-D. Tran, N. Pal, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, “Verification of piecewise deep neural networks: a star set approach with zonotope pre-filter,” *Formal Aspects of Computing*, vol. 33, no. 4, pp. 519–545, 2021.

[40] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh, “Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers,” *arXiv preprint arXiv:2011.13824*, 2020.

[41] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter, “Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification,” *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[42] V. R. Royo, R. Calandra, D. M. Stipanovic, and C. Tomlin, “Fast neural network verification via shadow prices,” *arXiv preprint arXiv:1902.07247*, 2019.

[43] F. Jaeckle, J. Lu, and M. P. Kumar, “Neural network branch-and-bound for neural network verification,” *arXiv preprint arXiv:2107.12855*, 2021.

[44] B. Paulsen, J. Wang, and C. Wang, “Reludiff: Differential verification of deep neural networks,” *arXiv preprint arXiv:2001.03662*, 2020.

[45] R. Bunel, P. Mudigonda, I. Turkaslan, P. Torr, J. Lu, and P. Kohli, “Branch and bound for piecewise linear neural network verification,” *Journal of Machine Learning Research*, vol. 21, no. 2020, 2020.

[46] A. De Palma, R. Bunel, A. Desmaison, K. Dvijotham, P. Kohli, P. H. Torr, and M. P. Kumar, “Improved branch and bound for neural network verification via lagrangian decomposition,” *arXiv preprint arXiv:2104.06718*, 2021.

[47] J. Li, J. Liu, P. Yang, L. Chen, X. Huang, and L. Zhang, “Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification,” in *International Static Analysis Symposium*. Springer, 2019, pp. 296–319.

[48] G. Singh, T. Gehr, M. Püschel, and M. Vechev, “Boosting robustness certification of neural networks,” in *International conference on learning representations*, 2018.

[49] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, “Efficient neural network robustness certification with general activation functions,” *Advances in Neural Information Processing Systems*, vol. 31, pp. 4939–4948, 2018.

[50] M. N. Müller, G. Makarchuk, G. Singh, M. Püschel, and M. Vechev, “Prima: general and precise neural network certification via scalable convex hull approximations,” *Proceedings of the ACM on Programming Languages*, vol. 6, no. POPL, pp. 1–33, 2022.