Towards Efficient Verification of Quantized Neural Networks

Pei Huang[1], Haoze Wu[1], Yuting Yang[2], Ieva Daukantas[3], Min Wu[1], Yedi Zhang[4], Clark Barrett[1]*

[1]Stanford University, Stanford, USA
[2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[3] IT University of Copenhagen, Denmark
[4] National University of Singapore, Singapore
{huangpei,haozewu,minwu}@stanford.edu, yangyuting@ict.ac.cn
daukantas@itu.dk, yd.zhang@nus.edu.sg, barrett@cs.stanford.edu

Abstract

Quantization replaces floating point arithmetic with integer arithmetic in deep neural network models, providing more efficient on-device inference with less power and memory. In this work, we propose a framework for formally *verifying* properties of quantized neural networks. Our baseline technique is based on integer linear programming which guarantees both *soundness* and *completeness*. We then show how efficiency can be improved by utilizing gradient-based heuristic search methods and also bound-propagation techniques. We evaluate our approach on perception networks quantized with PyTorch. Our results show that we can verify quantized networks with better scalability and efficiency than the previous state of the art.

Introduction

In recent years, deep neural networks (DNNs) (Goodfellow, Bengio, and Courville 2016) have demonstrated tremendous capabilities across a wide range of tasks (Simonyan and Zisserman 2015; Devlin et al. 2019; Dosovitskiy et al. 2021). However, DNNs have also shown various security and safety issues, e.g., vulnerability to input perturbations (Goodfellow, Shlens, and Szegedy 2015; Huang et al. 2022b,a; Yang et al. 2023). Such issues must be addressed before DNNs can be used in safety-critical scenarios such as autonomous driving (Xu et al. 2017) and medical diagnostics (Ciresan et al. 2012). Formal verification is an established technique which applies mathematical reasoning to ensure the correct behavior of safety-critical systems, and several approaches for applying formal methods to DNNs have been investigated (Huang et al. 2017; Lechner et al. 2022).

Our focus is the verification of quantized neural networks (QNNs). Quantization replaces inputs and parameters represented as 32/64-bit floating point numbers with a lower bit-width fixed point (e.g., 8-bits) representation (Jacob et al. 2018; Han, Mao, and Dally 2016). QNNs can greatly reduce both memory requirements and computational costs while maintaining competitive accuracy. As a result, they are increasingly being used in embedded applications, including safety-critical applications such as autonomous driving. For instance, 8-bit quantized DNNs have been applied in Tesla's Full Self-Driving Chip (previously Autopilot Hardware 3.0) (Henzinger, Lechner, and Žikelić 2021; Tes FSD Chip-Tesla). With the increasing popularization and use of QNNs, it is urgent to develop efficient and effective verification techniques for them.

In this work, we propose an efficient verification framework for QNNs with three components, offering different trade-offs between scalability and precision. The baseline approach models neural networks and formal properties as integer linear programming (ILP) problems. ILP is an exact method in the sense that it guarantees both *soundness* (if it reports the system is safe, then it really is safe) and *completeness* (if the system really is safe, then it will report that it is safe). Unlike previous work, which focuses on simple models of quantized neural networks, ours is the first formal approach that precisely captures the quantization scheme used in popular deep learning frameworks such as PyTorch/TensorFlow.

Our ILP approach is precise but may encounter scalability issues on larger QNNs. To address this, we also propose a gradient-based method for finding counterexamples. We use a rewriting trick for the non-differentiable round operation, which enables the backward process to cross through the round operation and gives us the desired gradient information. If this method finds a counterexample, then we immediately know that the property does not hold, without having to invoke the ILP solver.

The third component of the framework lies in between the first two. It relies on abstract interpretation-based reasoning to do an incomplete but formal analysis. We extend existing abstract interpretation-based interval analysis methods to support the semantics of "round" and "clip" operations in quantized neural networks. In particular, for the clip operation, we reduce it to a gadget built from two $ReLU$ units. If the abstract interpretation approach succeeds, we know the property holds. Otherwise, the result of the analysis can be used to reduce the runtime of the ILP-based complete method. The overall framework is depicted in Fig. 1.

Based on our framework, we realize an **E**fficient **Q**NN **V**erification system named EQV. We use EQV to verify the robustness of QNNs against bounded input perturbations. Our experimental results show that EQV can scale to networks that are more than twice as large as the largest

*Corresponding Author.

networks handled by previous approaches. We also show that, compared to the baseline ILP technique, EQV is up to 100 × more efficient for some cases. Our contributions can be summarized as the following: (1) We provide a ILP-based exact verification approach for the QNNs which first precisely captures the quantization scheme used in current popular deep learning frameworks; (2) We extend existing abstract interpretation-based interval analysis methods to support QNNs; (3) We design a rewriting trick for the non-differentiable round operation, which enables gradient-based analysis of QNNs; (4) We implement our approach in a tool, EQV, and demonstrate that it can scale to networks that are twice the size of the largest analyzed by the current state-of-the-art methods, and up to 100 × faster than the baseline ILP method.

Background and Related Work

Formal DNN verification checks whether a DNN satisfies a property such as the absence of adversarial examples in a given perturbation space. The property is usually depicted by a formal specification, and verifiers aim to provide either a proof of the validity of this property or a counterexample. Researchers have developed a range of verification techniques, mostly for real-valued $ReLU$ networks. Exact methods (i.e., sound and complete) can always, in theory, answer whether a property holds or not in any situation. Typical exact methods formalize the verification problem as a Satisfiability Modulo Theories (SMT) problem (Katz et al. 2017; Ehlers 2017; Huang et al. 2017; Jia et al. 2023) or a Mixed Integer Linear Programming (MILP) problem (Cheng, Nührenberg, and Ruess 2017; Fischetti and Jo 2018; Dutta et al. 2018), but their scalability is limited as the problem is NP-hard (Katz et al. 2017). Another typical approach is to use a method that only guarantees soundness, i.e., to improve the scalability at the cost of completeness. Abstract interpretation is one such approach. It overapproximates the behavior of the neural network with the hope that the property can still be shown to hold (Wong and Kolter 2018; Weng et al. 2018; Gehr et al. 2018; Zhang et al. 2018; Raghunathan, Steinhardt, and Liang 2018; Mirman, Gehr, and Vechev 2018; Singh et al. 2019). Finally, heuristic approaches can be used to search for counterexamples. These techniques are neither sound nor complete but can be effective in practice (Goodfellow, Shlens, and Szegedy 2015; Yang et al. 2022; Serban, Poll, and Visser 2021).

Existing work on DNN verification typically focuses on networks whose parameters are real or floating point numbers. In contrast, relatively little prior work addresses the verification of QNNs. QNN verification presents additional challenges due to the difficulty of modeling quantization schemes. And some evidence suggests that it may also be more computationally challenging. For example, Jia et al. (Jia and Rinard 2020) point out that the verification of binarized neural networks (which can be regarded as 1-bit quantized neural networks) has exhibited even worse scalability than real-valued neural network verification.

In the last two years, some work has started to focus on the verification of QNNs. Henzinger et al. (Henzinger, Lechner, and Žikelić 2021) provide an SMT-based method to encode the problem as a formula in the SMT theory of bit-vectors. Mistry et al. (Mistry, Saha, and Biswas 2022) and hang et al. (Zhang et al. 2022) propose using MILP and ILP to model the QNN verification problem. All of these methods pioneer new directions for QNNs but are applied only to small models using simple quantization schemes. None of them can directly support the sophisticated quantization schemes used in real deep learning frameworks.

Preliminaries

The quantization operation is a mapping from a real number γ to an integer q of the form

$$\text{Quant: } q = Round(\frac{\gamma}{s} + z), \text{ De-quant: } \gamma = s(q - z), \quad (1)$$

for some constants s and z. Equation 1 is the quantization scheme, and the constants s and z are quantization parameters. The constant s (for "scale") is an arbitrary real number. The constant z (for "zero point") is the integer corresponding to the quantized value q when $\gamma = 0$. In practice, q is represented using a fixed number of bits. For example, in 8-bit quantization, q is an 8-bit integer. Note that in general, q may not fit within the number of bits provided, in which case the closest representable value is used.

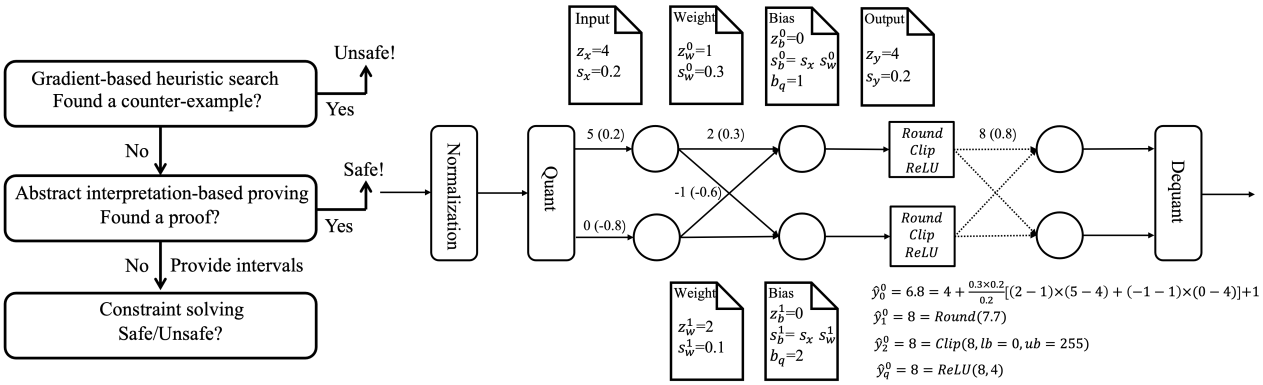
One of the most important operations when doing forward inference in DNNs is matrix multiplication. Suppose we have three $N \times N$ matrices of real numbers, where the third matrix is equal to the product of the first two matrices. Denote the entries of these 3 matrices as $r_\alpha^{(i,j)}$, where $\alpha \in \{1, 2, 3\}$ and $0 \leq i, j \leq N - 1$. Their quantization parameters are (s_α, z_α) (in general, different quantization parameters may be used for different neurons in a DNN). We use $q_\alpha^{(i,j)}$ to denote the quantized entries of these 3 matrices. Based on the quantization scheme $r_\alpha^{(i,j)} = s_\alpha(q_\alpha^{(i,j)} - z_\alpha)$ and the definition of matrix multiplication, we have

$$s_3(q_3^{(i,j)} - z_3) = \sum_{k=0}^{N-1} s_1(q_1^{(i,k)} - z_1)s_2(q_2^{(k,j)} - z_2), \quad (2)$$

which can be rewritten as

$$q_3^{(i,j)} = z_3 + \frac{s_1 s_2}{s_3} \sum_{k=0}^{N-1} (q_1^{(i,k)} - z_1)(q_2^{(k,j)} - z_2). \quad (3)$$

Suppose $\mathbf{y} := ReLU(W\mathbf{x} + \mathbf{b})$ is the function describing the transformation performed in a single layer of a DNN. Its quantized version $\mathbf{y}_q := g(\mathbf{x}_q, W_q, \mathbf{b}_q)$ can be described by the series of calculations shown in Equation (4), where W_q, $\mathbf{b}_q$, $\mathbf{x}_q$ and $\mathbf{y}_q$ are the quantized versions of the weight matrix W, bias vector $\mathbf{b}$, input vector $\mathbf{x}$, and output vector $\mathbf{y}$, respectively. Let $z_\mathbf{x}$ and $z_\mathbf{y}$ be the zero points of $\mathbf{x}$ and $\mathbf{y}$ respectively. As the zero points of the weights corresponding to each output neuron may be different, we use z_w^j to denote the zero point of the weights corresponding to the j-th neuron. Similarly, s_w^j, $s_\mathbf{x}$, and $s_\mathbf{y}$ are the scales for the weight matrix, input, and output, respectively. The $ReLU$ function in the quantized network can be represented as the



(a) Main verification framework. (b) Quantized neural network. A number in "( )" is its corresponding fixed-point representation.

Figure 1: Framework and QNN

maximum of the input and the zero point. The calculation for $\mathbf{y}_q := g(\mathbf{x}_q, W_q, \mathbf{b}_q)$ can then be written as:

$$\begin{aligned}
&\text{(i) } \hat{y}_0^j := z_y + \frac{s_w^j s_x}{s_y} \sum_i (w_q^{(i,j)} - z_w^j)(x_q^i - z_x) + b_q^j \\
&\text{(ii) } \hat{y}_1^j := Round(\hat{y}_0^j) \\
&\text{(iii) } \hat{y}_2^j := Clip(\hat{y}_1^j, lb, ub) \\
&\text{(iv) } y_q^j := \max(\hat{y}_2^j, z_y)
\end{aligned} \tag{4}$$

where lb and ub are the smallest and largest values, respectively, that can be represented by our quantized integer type, e.g., for an 8-bit unsigned type, $[lb, ub]$=$[0, 255]$. The $Clip$ function returns the value within $[lb, ub]$ closest to its input. In Pytorch, weights are usually quantized as signed integers while the inputs and outputs of each layer are quantized as unsigned integers. The quantization parameters (i.e., zero points and scales) are computed offline and determined at the time of quantization. In the inference phase, they are constants. Fig. 1 shows a QNN performing an example computation.

The property utilized in this paper for testing the verification efficiency is robustness. Let $f : \mathbb{D}^n \to \mathbb{O}^m$ be a neural network classifier, where, for a given input $x \in \mathbb{D}^n$, $f(x) = \{o_0(x), o_1(x), ..., o_{m-1}(x)\} \in \mathbb{O}^m$ represents the confidence values for m classification labels. In general $\mathbb{D}$ and $\mathbb{O}$ are sets of real numbers, and for quantized neural networks they are sets of integers corresponding to the quantization type. The prediction of x is given as $F(x) = \arg\max_{0 \le i \le m-1} o_i(x)$, and the label space is denoted as $\mathcal{Y}$. The robustness property can be depicted as: given a test point x_* with label l_*, a neural network is locally robust at point x_* with respect to a perturbation radius r if the following formula holds:

$$\forall x \; (x \in B_\infty(x_*, r) \to F(x) = l_*) \tag{5}$$

where $B_\infty(x_*, r) = \{x \mid \|x - x_*\|_\infty \le r\}$ is the perturbation space around x_* bounded by an ℓ_∞-norm ball of radius r. The goal of the verifier is to answer whether Equation (5) holds.

ILP Modeling

In this section, we introduce an ILP formulation for the QNN robustness verification problem. Compared with previous work on QNN verification (Mistry, Saha, and Biswas 2022; Zhang et al. 2022), the main difference is that our encoding correctly models quantization schemes used in mainstream deep learning frameworks (e.g., PyTorch). In addition, unlike (Mistry, Saha, and Biswas 2022), we avoid using floating point variables, as in our experience, it is easier to solve ILP problems than to solve MILP problems. And in contrast to (Zhang et al. 2022), we avoid piecewise constraints which introduce many redundant variables.

In this paper, we use a symbol with a dot ("$\cdot$") to denote a variable in our ILP model corresponding to an input to output from some layer of the DNN, e.g., variable $\dot{y}$.

We show how to encode each step in calculation (4). For step (i), we use the variable $\dot{x}_q^i$ for the i-th component of the input and an auxiliary variable $\hat{y}_0^j$ to denote the result. Note that $\hat{y}_0^j$ is a temporary variable and is not of integer type. The introduction of this symbol is for the sake of convenience, and we show how to eliminate it below.

$$\hat{y}_0^j = z_y + \frac{s_w^j s_x}{s_y} \sum_i (w_q^{(i,j)} - z_w^j)(\dot{x}_q^i - z_x) + b_q^j \tag{6}$$

For step (ii), $\hat{y}_1^j := Round(\hat{y}_0^j)$ can be encoded by the following two constraints:

$$\begin{cases} \dot{\hat{y}}_1^j - \hat{y}_0^j \le 0.5 \\ \hat{y}_0^j - \dot{\hat{y}}_1^j \le 0.5 - \varepsilon, \end{cases} \tag{7}$$

where a small constant ε is used to realize the "$<$" operator (in ILP solvers, this operator is usually not supported directly). Since the result of the sum in Equation (6) is always an integer, we can find a proper value for ε based on the factor $s_w^j s_x / s_y$ which guarantees the correctness of the encoding.

We now eliminate the temporary variable $\hat{y}_0^j$ by combin-

ing constraints (6) and (7):

$$\begin{cases} \dot{\hat{y}}_1^j - z_y - \frac{s_w^j s_x}{s_y} \sum_i (w_q^{(i,j)} - z_w^j)(\dot{x}_q^i - z_x) - b_q^j \leq 0.5 \\ z_y + \frac{s_w^j s_x}{s_y} \sum_i (w_q^{(i,j)} - z_w^j)(\dot{x}_q^i - z_x) + b_q^j - \dot{\hat{y}}_1^j \leq 0.5\text{-}\varepsilon \end{cases} \tag{8}$$

Let $Encode\_max(z, x, y)$ denote the ILP encoding of $z = \max(x, y)$ which can be realized with big M method (Cheng, Nührenberg, and Ruess 2017):

$$Encode\_max(z, x, y) = \begin{cases} b_x + b_y = 1 \\ x - z \leq 0 \\ y - z \leq 0 \\ x - z + Mb_y \geq 0 \\ y - z + Mb_x \geq 0 \\ y - x + Mb_x \geq 0 \end{cases} \tag{9}$$

where M is a very large positive constant and b_x, b_y are fresh 0-1 type integer variables. It is worth noting that we use this same encoding even if one of x and y is a constant value.

For step (iii) $\hat{y}_2^j := Clip(\hat{y}_1^j, lb, ub)$, the constraints are:

$$Encode\_max(\dot{\hat{y}}_{max}^j, \dot{\hat{y}}_1^j, lb) \bigcup Encode\_min(\dot{\hat{y}}_2^j, \dot{\hat{y}}_{max}^j, ub) \tag{10}$$

where $\dot{\hat{y}}_{max}^j$ is a fresh auxiliary variable. Finally, step (iv) can be directly written as $Encode\_max(\dot{\hat{y}}_q^j, \dot{\hat{y}}_2^j, z_y)$.

In our ILP model, the input variables represent the values of the input after input quantization. So the upper and lower bounds of the perturbation space also need to be quantized when representing the input constraint.

Encoding for Typical Fusion Layers

In order to reduce the amount of computation required for a quantized neural network during inference, certain layers are fused by the quantization process so that one kernel call does the computation for several neural network layers. We can use the same approach to reduce the number of constraints and variables in our ILP encoding.

Fusion of affine transformations and batch normalization In the inference phase, the parameters of a batch normalization layer are fixed, making it an affine transformation, e.g., $\mathbf{y} = BN(\mathbf{x}) = \gamma(\mathbf{x} - \mu_\mathbf{x})/\sqrt{\sigma_x^2 + \epsilon} + \beta$. Two consecutive affine transformations can always be rewritten as a new single affine transformation. For example, $\mathbf{y} = BN(W\mathbf{x} + \mathbf{b})$ can be regarded as a new affine transformation $\mathbf{y} = W'\mathbf{x} + \mathbf{b}'$. Therefore, a convolutional layer (or a linear layer) can be fused with a batch normalization layer to get a new convolutional layer (or linear layer), where the input tensor $\mathbf{x}$ remains unchanged but the weight and bias parameters are updated accordingly. In our ILP encoding, we also fuse such layers. The encoding process is the same, but using the modified weights and biases.

Fusion of affine transformations and ReLU The $Clip$ operation has a similar function to $ReLU$: they both limit the lower bound of the output. So, in the quantification process, these two operations can be fused together into one $Clip$ operation. For example, in a quantized neural network, the convolutional and $ReLU$ layers can be merged together to form a $ConvReLU$ layer. For our encoding, we combine steps (iii) and (iv) to get a single $Clip$ operation:

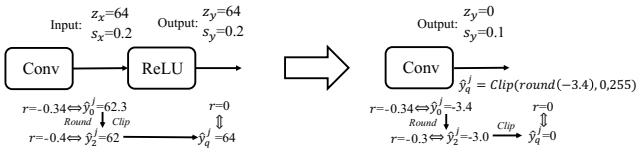


Figure 2: Fusion of the convolutional layer and $ReLU$.

$$\text{(iii} \oplus \text{iv)}\ y_q^j := Clip(\hat{y}_1^j, lb', ub), \tag{11}$$

where lb' is the new lower bound. Fig. 2 shows an example. We can replace lb and $\hat{y}_2^j$ in constraint (10) with lb' and $\hat{y}_q^j$ to generate the encoding for (iii $\oplus$ iv).

Interval Analysis

In the second step of our verification framework, we compute the lower bound $lb(\dot{y})$ and upper bound $ub(\dot{y})$ for each variable $\dot{y}$. After obtaining the bounds for each variable, sometimes we can directly conclude that the property holds. Even if the bounds are not precise enough to prove the property, we can use them to simplify the ILP problem. In particular, the bounds may be able to show that some neurons are always active or always inactive. To compute the bounds, we use standard abstract interpretation techniques which use convex polyhedra to over-estimate the output interval of each node (Wang et al. 2018). We make two small contributions in this context that help support our goal of verifying QNNs.

First, we add support for the round operation, $\hat{y}_1^j := Round(\hat{y}_0^j)$. For this operation, the bounds on the output can easily be determined from the bounds on the input based on (7), which can be rewritten as:

$$\varepsilon - 0.5 + \hat{y}_0^j \leq \hat{y}_1^j \leq 0.5 + \hat{y}_0^j \tag{12}$$

The other contribution is to support the clip operation, $\hat{y}_2^j := Clip(\hat{y}_1^j, lb, ub)$. Our solution is to use $ReLU$ to simulate its function. The advantage of this method is that we can then leverage abstract interpretation techniques for $ReLU$, which have been extensively studied and optimized (Singh et al. 2019; Wu and Zhang 2021). The expression is:

$$\hat{y}_{max}^j := ReLU(\hat{y}_1^j, lb), \hat{y}_2^j := ub - ReLU(ub - \hat{y}_{max}^j) \tag{13}$$

In other words, we can add the following structure (Fig. 3) to the network and then use existing techniques to compute the bounds.

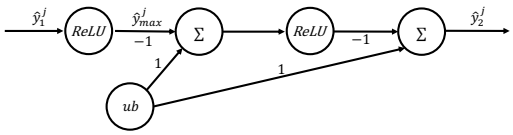


Figure 3: Using two $ReLU$s to simulate $Clip(\cdot)$.

We implemented these two techniques in the Marabou neural network verification tool (Katz et al. 2019) which has support for abstract interpretation-based bound computation. This allows us to use Marabou to compute bounds for our QNNs.

Gradient-based Heuristic Search

Heuristic search can sometimes be very efficient at finding counterexamples to formal properties. We can thus use it as a complementary approach to abstract interpretation and exact verification. In particular, gradient-based heuristic search is an effective technique. However, in popular implementations of quantization schemes (e.g., PyTorch), the gradients of QNNs are unavailable. One possible solution is to construct a new dummy neural network just for the purposes of gradient computation. In the dummy network, we copy the structure and weights from the QNN but use the full floating-point representation. However, there is still an issue that must be addressed, which is that the $Round$ function is not differentiable. Indeed, it has derivative 0 at all points except where it is discontinuous. However, if we look at the overall direction that the $Round$ function moves as we increase or decrease the input, it never goes too far away from $y = x$. This suggests that using $y = x$ in place of $y = Round(x)$ may be a good approximation.

For example, if we have $y = Round(y_1)$ and $y_1 = f(x)$, where f is a differentiable function, we would like for $\partial y/\partial x$ to simply be computed as $\partial y_1/\partial x$. To do so, the $Round$ operation can simply be omitted and we can just let $y = f(x)$ and get its gradient. However, this naive approach does not work when there are multiple layers with $Round$ operations. Suppose we have $y = Round(y_3)$, $y_3 = f_2(y_2)$, $y_2 = Round(y_1)$, $y_1 = f(x)$, and we compute the gradient as follows:

$$\frac{\partial y}{\partial x} = \frac{\partial y_3}{\partial y_2}\frac{\partial y_1}{\partial x}.$$

We can see that the gradient value is related to the value of y_2 (the output of the $Round$ operation). But we have essentially dropped this value, so if we just remove all the $Round$ operations from the quantized neural network, this will cause an accumulation in the gradient error layer by layer.

To resolve this problem, we use a trick to rewrite the $Round$ operation in the dummy network so that both the output value and the gradient are available. For each $Round(\cdot)$ operation in our dummy network, we replace it with

$$y = Round(x) + x - x.detach() \tag{14}$$

where $x.detach()$ denotes the operation that returns a new tensor with the same value as x but detached from the current computational graph. It is easy to see that the gradient of terms $Round(x)$ and $x.detach()$ are 0. So the value of the partial derivative of y with respect to x is the same as with the function $y = x$. Once we construct the dummy network, we can use a standard gradient-based attack to find counterexamples. In our experiments, our implementation is similar to the PGD algorithm (Madry et al. 2018).

		FC1-100	FC2-100
$r = 4$	QVIP	11751.16 (30)	30000.00(100)
	ILP	332.48 (0)	4039.38 (7)
	ILP+In	**93.52 (0)**	1581.41 (3)
	EQV	104.09 (0)	**1465.46 (3)**
$r = 8$	QVIP	30000.00 (100)	30000.00(100)
	ILP	2598.41 (7)	25121.16 (77)
	ILP+In	2729.48 (7)	13854.10 (38)
	EQV	**1463.89 (2)**	**12064.63 (32)**
$r = 12$	QVIP	29512.08 (98)	30000.00(100)
	ILP	10640.24 (31)	29589.61 (98)
	ILP+In	13467.12 (36)	26069.18 (81)
	EQV	**7726.74 (17)**	**23427.27 (73)**
$r = 16$	QVIP	28945.63 (96)	30000.00(100)
	ILP	18925.01 (57)	29750.82 (98)
	ILP+In	21172.09 (60)	28902.99 (94)
	EQV	**6621.84 (15)**	**22724.67 (73)**

Table 1: Comparisons between QVIP and EQV.

Experiments

We implemented a Python-based verification tool called EQV[1]. We use Gurobi (Gurobi 2018) as our backend ILP solver and the abstract interpretation is done by Marabou. The implementation of heuristic search and some results are described in the Appendix[2]. The efficiency of our approach is evaluated on two well-known neural network architectures: fully connected neural networks (FC) and convolutional neural networks (CNN). We use the notation FCN-M to refer to a network consisting of N dense layers with M hidden units in each layer. For example, the structure of FC2-256 is 784 $\times$ 256$\times$ 256 $\times$ 10. CNN1 is a network with one convolutional layer of 4 channels, followed by one batch normalization layer, one max-pooling layer with a kernel size of 2, and a fully connected layer with 10 units. The convolutional layer has 4 $\times$ 4 filters and 2 $\times$ 2 strides with a padding of 1. CNN2 is identical to CNN1 except that its convolutional layer has 2 channels. All neural networks are trained on the MNIST dataset (LeCun et al. 1998) and quantized with PyTorch using its default static quantization scheme. Verification experiments are conducted on the test set. In particular, we verify the robustness of networks (using Equation (5)) with $r = 4, 8, 12, 16$. The experimental environment is a 20-core Intel(R) Xeon(R) E5-2640 v4 @ 2.40GHz CPU with 64GB of memory.

Comparison To show the effectiveness of our strategies, three variants of our method: pure ILP, ILP with abstract interpretation (ILP+In), and EQV are involved in the experiments. We first compare with the SOTA QNN verification tool QVIP (Zhang et al. 2022) in terms of efficiency. What must be stated is that QVIP only supports a simplified quantization scheme and does not support the quantization scheme used in PyTorch. Although the network architecture used in the experiment is the same, the weights and computational processes of the network are not entirely identical. The size of FC2-100 has exceeded the maximum network size used in the experiment for QVIP. This experiment is

[1]https://github.com/huangdiudiu/EQV

[2]Appendix is in http://arxiv.org/abs/2312.12679.

		FC2-256	FC2-512	FC3-100	CNN1	CNN2
$r = 4$	ILP	22111.22 (67)	29835.94 (97)	29183.30 (95)	28235.35 (94)	28015.32 (93)
	ILP+In	4456.76 (11)	8013.00 (16)	3347.88 (8)	221.64 (0)	1041.85 (2)
	EQV	**4019.10 (10)**	**7637.05 (15)**	**2421.66 (5)**	**205.48 (0)**	**723.13 (1)**
$r = 8$	ILP	29991.71 (99)	30000.00 (100)	30000.00 (100)	25345.75 (84)	27357.55 (90)
	ILP+In	22543.68 (62)	29891.39 (89)	24330.21 (78)	2503.19 (5)	3715.99 (9)
	EQV	**19918.79 (55)**	**29179.91 (87)**	**22616.44 (72)**	**1850.32 (4)**	**1499.73 (3)**
$r = 12$	ILP	29660.15 (99)	30000.00 (100)	29901.57 (99)	21609.24 (70)	27422.63 (90)
	ILP+In	29991.41 (99)	29930.07 (99)	29559.77 (96)	5236.63 (11)	13949.19 (38)
	EQV	**23938.37 (77)**	**29362.36 (92)**	**24137.56 (78)**	**4698.57 (11)**	**8333.80 (24)**
$r = 16$	ILP	29991.33 (99)	30000.00 (100)	29919.99 (99)	15004.02 (46)	28330.85 (93)
	ILP+In	29767.65 (99)	30000.00 (100)	29928.29 (99)	5084.85 (11)	18977.19 (55)
	ILP+In+PGD	**12021.24 (50)**	**24647.06 (78)**	**16402.12 (54)**	**3897.42 (10)**	**6733.25(21)**

Table 2: Total execution time(s) of different methods.

only intended for reference. The comparison between our methods and EQV is shown in Table 1. The number in "()" indicates the number of timeouts (300s). We randomly select the 100 examples from the test set, and any instance that exceeds the timeout is recorded as 300 seconds.

Table 1 demonstrates that EQV outperforms QVIP largely with less time and fewer timeouts. Our pure ILP method has achieved efficiency improvements of several tens of times compared to QVIP when the radius is small (e.g.$r = 4, 8$). Especially for FC-100, our methods improve efficiency by up to 78 times. Although the quantization scheme of the networks differs, comparisons between ILP and ILP+In against QVIP reveal that avoiding the use of piecewise constraints can significantly improve the verification efficiency.

Performances on Larger NNs Table 2 shows the performance of our methods on some larger neural networks. From Table 2, we can see that: (1) abstract interpretation excels at handling cases with small radii, while heuristic search excels at finding counterexamples with large radii, confirming that these techniques are complementary; (2) in some cases, ILP+In is slower than ILP, indicating that abstract interpretation is not well-suited for these cases and simply adds overhead; however, when we add heuristic search, efficiency is improved, suggesting that heuristic search can sometimes compensate for the efficiency reduction caused by "In"; (3) for the same network, the efficiency of EQV decreases initially and then increases with the growth of the radius; the most challenging cases appear to occur near the maximum safe radius; (4) For some cases where the radius is less than 4, EQV is more than 100 $\times$ faster than the baseline.

Fig. 4 provides a comprehensive assessment of the three methods across the entire set of solved instances, encompassing all seven networks. The X-axis represents the number of solved instances and the Y-axis represents the time needed to solve each of them. We can observe that EQV consistently demonstrates superior performance, and as the perturbation radius increases, its efficiency advantage over other methods becomes more pronounced.

The Effectiveness of Different Parts In the experiment, we recorded the contributions of each component of EQV. Taking FC-100 as an example, Figure 5 shows the percentage of instances that were solved by each component. When the radius is relatively small, bound propagation primarily plays a major role in accelerating the solving of many instances; when the radius takes intermediate values, bound propagation can provide tighter bounds for ILP variables, thus expediting the solving process; when the radius is relatively large, bound propagation becomes ineffective, and heuristic search can compensate for efficiency loss by rapidly identifying counterexamples.

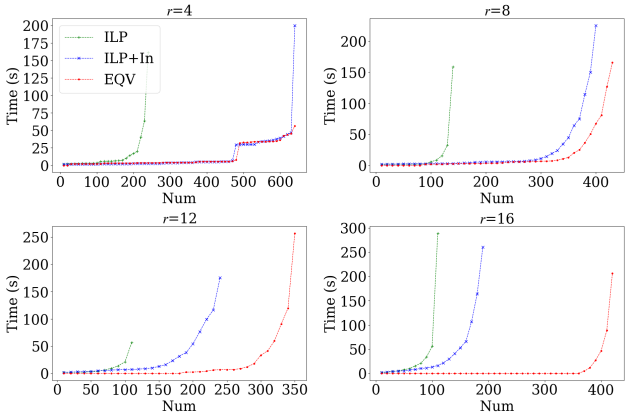


Figure 4: Performance curve on all the instances.

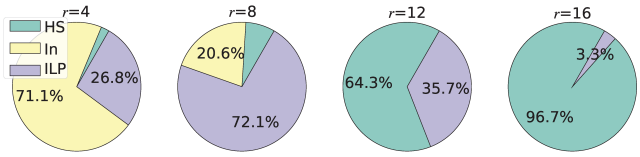


Figure 5: The percentage of examples solved by each strategy (HS: heuristic search).

Efficiency of Parallel Methods Based on Tables 2 and Tables 3, it is clear that for larger networks and values of r, (e.g., for FC2-100, FC2-256 when $r = 4, 8, 12, 16$), the problems become quite challenging for EQV. We did a preliminary investigation to determine whether parallel solving can help in these instances. We ran the ILP and abstract interpretation methods with 20 parallel threads. Taking it a step further, to explore the impact of increased solving time on the success rate of verification, we also conducted ex-

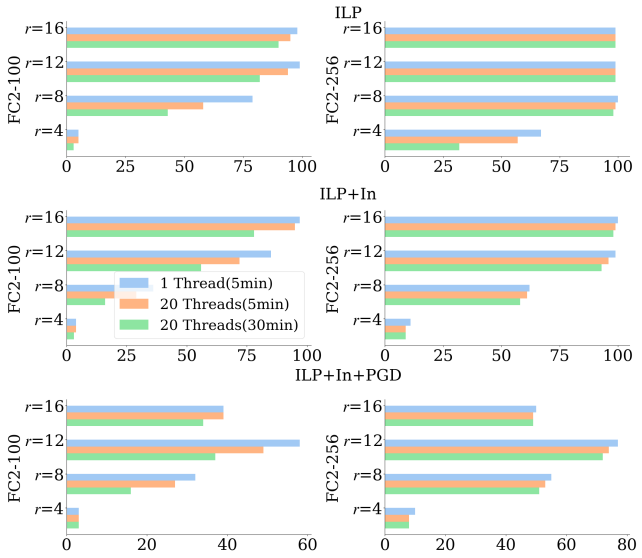


Figure 6: The number of unknown instances under various configurations.

periments with a time limit of 30 minutes for each instance. The results are presented in Fig 6. Parallel algorithms do improve efficiency, but their improvements are less than 10%. This indicates that a straightforward parallel approach has limited efficiency gains, and we need to explore parallel algorithms tailored for neural networks. There is still much room for improvement in parallel verification methods. We also observe that with extended time limits, the ILP method can sometimes solve nearly 50% of the instances, but for EQV, providing more time yields only marginal improvements. This indicates that the complementary acceleration strategies employed within EQV have significantly leveraged the potential for accelerating the verification tool. Our acceleration strategy's effectiveness can be observed from Fig. 6, even surpassing the parallel acceleration for the basic method with 20 threads.

Verification Results Table 3 shows the results given by EQV within 30 minutes. Acc is the accuracy of the QNN; "Rob" is the percentage of instances proven robust by EQV; "Uns" is the percentage shown to be unsafe by EQV; and "Unk" is the percentage that is unknown (i.e. timeouts). Notice that when $r = 4$, even for the largest network (FC2-512), our method provides an answer for nearly 88% of the instances within the timeout. This again suggests greater scalability than previous approaches.

Robustness Changes Caused by Quantization To investigate the effect of quantization on the robustness of neural networks, we compare the robustness verification results of the original networks with those of the quantized neural networks. The experiments were conducted on 4 networks, namely FC1-100, FC2-100, CNN1 and CNN2. Fig. 7 plots the percentage of instances shown to be safe at each radius. The curve of "safe+unknown" can be regarded as the success rate of resisting attacks. For QNN, the gap between the

r		1-100	2-100	2-256	2-512	3-100	CNN1	CNN2
4	Rob	96	95	90	88	93	90	89
	Uns	2	2	2	1	3	8	7
	Unk	0	3	8	11	4	0	0
8	Rob	89	79	41	16	45	64	75
	Uns	9	5	8	2	6	33	20
	Unk	0	16	51	82	49	1	1
12	Rob	62	36	6	1	12	31	43
	Uns	28	27	22	7	18	64	46
	Unk	8	37	72	92	70	3	7
16	Rob	33	7	1	0	0	11	10
	Uns	57	59	50	22	46	87	78
	Unk	8	34	49	78	54	0	8

Table 3: Results given by EQV (30 minutes).

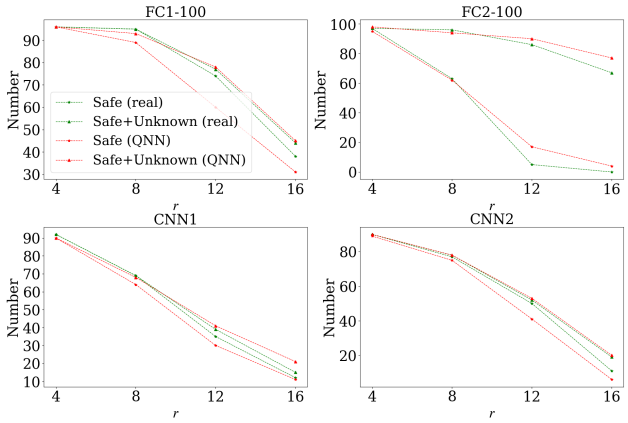


Figure 7: Robustness curves of NN and QNN.

two curves "safe+unknown" and "safe" is larger than that of the real-valued network, indicating that the verification complexity of QNN might be greater than verifying a real-valued network. It is also interesting to observe that although quantization reduces the accuracy of a neural network, it does not always reduce the robustness of the network. From the point of view of resisting attack, when the radius is below a certain threshold, the real-valued networks exhibit better robustness, whereas when the radius exceeds a certain threshold, QNNs demonstrate superior robustness.

Conclusion

In this work, we propose an efficient verification framework for QNNs that offers different trade-offs between scalability and precision. Our verification tool EQV is the first formal verification tool that precisely captures the quantization scheme used in popular deep learning frameworks. Although we focus on verifying adversarial robustness, our method could be generalized to verify other properties of QNNs. Experimental results show that EQV is more efficient and scalable than previously existing approaches. In future work, it would be interesting to formally analyze the difference or equivalence between the original networks and the quantized neural networks or to formally quantify the precision loss due to the quantization process.

Acknowledgments

This work was funded in part by the Stanford Center for AI Safety and by a Ford Alliance Project (199909).

References

FSD Chip-Tesla. 2019. https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip. Accessed: 2019-04-30.

Cheng, C.; Nührenberg, G.; and Ruess, H. 2017. Maximum Resilience of Artificial Neural Networks. In *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, volume 10482 of *Lecture Notes in Computer Science*, 251–268. Springer.

Ciresan, D. C.; Giusti, A.; Gambardella, L. M.; and Schmidhuber, J. 2012. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, 2852–2860.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1*, 4171–4186. Association for Computational Linguistics.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Dutta, S.; Jha, S.; Sankaranarayanan, S.; and Tiwari, A. 2018. Output Range Analysis for Deep Feedforward Neural Networks. In *NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings*, volume 10811 of *Lecture Notes in Computer Science*, 121–138. Springer.

Ehlers, R. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, volume 10482 of *Lecture Notes in Computer Science*, 269–286. Springer.

Fischetti, M.; and Jo, J. 2018. Deep neural networks and mixed integer linear optimization. *Constraints An Int. J.*, 23(3): 296–309.

Gehr, T.; Mirman, M.; Drachsler-Cohen, D.; Tsankov, P.; Chaudhuri, S.; and Vechev, M. T. 2018. 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*, 3–18. IEEE Computer Society.

Goodfellow, I. J.; Bengio, Y.; and Courville, A. C. 2016. *Deep Learning*. Adaptive computation and machine learning. MIT Press.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Gurobi. 2018. A most powerful mathematical optimization solver.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Henzinger, T. A.; Lechner, M.; and Žikelić, D. 2021. Scalable verification of quantized neural networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, 3787–3795. AAAI Press.

Huang, P.; Yang, Y.; Jia, F.; Liu, M.; Ma, F.; and Zhang, J. 2022a. Word Level Robustness Enhancement: Fight Perturbation with Perturbation. 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*, 10785–10793. AAAI Press.

Huang, P.; Yang, Y.; Liu, M.; Jia, F.; Ma, F.; and Zhang, J. 2022b. ϵ-weakened robustness of deep neural networks. In *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*, 126–138. ACM.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety Verification of Deep Neural Networks. In *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*, 3–29. Springer.

Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A. G.; Adam, H.; and Kalenichenko, D. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 2704–2713. Computer Vision Foundation / IEEE Computer Society.

Jia, F.; Han, R.; Huang, P.; Liu, M.; Ma, F.; and Zhang, J. 2023. Improving Bit-Blasting for Nonlinear Integer Constraints. In Just, R.; and Fraser, G., eds., *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17-21, 2023*, 14–25. ACM.

Jia, K.; and Rinard, M. C. 2020. Efficient Exact Verification of Binarized Neural Networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Katz, G.; Barrett, C. W.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *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*, 97–117. Springer.

Katz, G.; Huang, D. A.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljic, A.; Dill, D. L.; Kochenderfer, M. J.; and Barrett, C. W. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, 443–452. Springer.

Lechner, M.; Zikelic, D.; Chatterjee, K.; and Henzinger, T. A. 2022. Stability Verification in Stochastic Control Systems via Neural Network Supermartingales. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Virtual Event, February 22 - March 1, 2022*, 7326–7336. AAAI Press.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11): 2278–2324.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Mirman, M.; Gehr, T.; and Vechev, M. T. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, 3575–3583. PMLR.

Mistry, S.; Saha, I.; and Biswas, S. 2022. An MILP Encoding for Efficient Verification of Quantized Deep Neural Networks. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 41(11): 4445–4456.

Raghunathan, A.; Steinhardt, J.; and Liang, P. 2018. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 10900–10910.

Serban, A. C.; Poll, E.; and Visser, J. 2021. Adversarial Examples on Object Recognition: A Comprehensive Survey. *ACM Comput. Surv.*, 53(3): 66:1–66:38.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. T. 2019. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL): 41:1–41:30.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018. Efficient Formal Safety Analysis of Neural Networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 6369–6379.

Weng, T.; Zhang, H.; Chen, H.; Song, Z.; Hsieh, C.; Daniel, L.; Boning, D. S.; and Dhillon, I. S. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, 5273–5282. PMLR.

Wong, E.; and Kolter, J. Z. 2018. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, 5283–5292. PMLR.

Wu, Y.; and Zhang, M. 2021. Tightening Robustness Verification of Convolutional Neural Networks with Fine-Grained Linear Approximation. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Event, February 2-9, 2021*, 11674–11681. AAAI Press.

Xu, H.; Gao, Y.; Yu, F.; and Darrell, T. 2017. End-to-End Learning of Driving Models from Large-Scale Video Datasets. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 3530–3538. IEEE Computer Society.

Yang, X.; Dong, Y.; Pang, T.; Su, H.; and Zhu, J. 2022. Boosting Transferability of Targeted Adversarial Examples via Hierarchical Generative Networks. In *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part IV*, volume 13664 of *Lecture Notes in Computer Science*, 725–742. Springer.

Yang, Y.; Huang, P.; Cao, J.; Ma, F.; Zhang, J.; and Li, J. 2023. Quantifying Robustness to Adversarial Word Substitutions. In *Machine Learning and Knowledge Discovery in Databases: Research Track - European Conference, ECML PKDD 2023, Turin, Italy, September 18-22, 2023, Proceedings, Part I*, volume 14169, 95–112. Springer.

Zhang, H.; Weng, T.; Chen, P.; Hsieh, C.; and Daniel, L. 2018. Efficient Neural Network Robustness Certification with General Activation Functions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 4944–4953.

Zhang, Y.; Zhao, Z.; Chen, G.; Song, F.; Zhang, M.; Chen, T.; and Sun, J. 2022. QVIP: An ILP-based Formal Verification Approach for Quantized Neural Networks. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*, 82:1–82:13. ACM.