Parallel Verification for δ-Equivalence of Neural Network Quantization

Pei Huang[1], Yuting Yang[2,3], Haoze Wu[1],
Ieva Daukantas[4], Min Wu[1], Fuqi Jia[1,2,5], and Clark Barrett[1]

[1] Stanford University, Stanford,CA, USA
[2] UCAS, Beijing, China
[3] Institute of Computing Technology, CAS, Beijing, China
[4] IT University of Copenhagen, Denmark
[5] Institute of Software, CAS, Beijing, China
{huangpei,haozewu,minwu,jiafq24,barrett}@stanford.edu
yangyuting@ict.ac.cn, daukantas@itu.dk

Abstract. *Quantization* replaces floating point arithmetic with integer arithmetic in deep neural networks, enabling more efficient on-device inference with less power and memory. However, it also brings in loss of generalization and even potential errors to the models. In this work, we propose a parallelization technique for formally *verifying* the *equivalence* between quantized models and their original real-valued counterparts. In order to guarantee both *soundness* and *completeness*, mixed integer linear programming (MILP) is deployed as the baseline technique. Nevertheless, the incorporation of two networks as well as the mixture of integer and real number arithmetic make the problem much more challenging than verifying a single network, and thus using MILP alone is inadequate for the non-trivial cases. To tackle this, we design a distributed verification technique that can leverage hundreds of CPUs on high-performance computing clusters. We develop a two-tier parallel framework and propose property- and output-based partition strategies. Evaluated on perception networks quantized with PyTorch, our approach outperforms existing methods in successfully verifying many cases that are otherwise considered infeasible.

Keywords: Quantized neural networks · Equivalence verification· Parallel computing.

1 Introduction

In the past few years, deep neural networks (DNNs) [11] have demonstrated striking and steadily improving capabilities across a wide range of tasks [39, 32, 5, 6]. Ongoing improvements to neural networks are typically achieved through a significant expansion in their scale. This escalation, however, frequently leads to substantial increases in computational cost, memory bandwidth requirements, and energy consumption, which make DNNs difficult to deploy on embedded systems with limited hardware resources. A promising approach for mitigating

this difficulty is *neural network quantization*. This technique replaces floating point arithmetic with integer arithmetic in deep models, enabling more efficient inference on devices with reduced power and memory requirements [21, 13]. Quantization has been employed, for example, in Tesla's FSD-chip [1] and in various mobile devices [25, 2]).

As quantization trades off precision for efficiency, the accuracy or generalizability of a quantized neural network (QNN) is typically reduced compared to that of the original real-valued network. Although the impact of neural network quantization can be measured through empirical indicators such as changes in accuracy on a test set, this is still insufficient, especially considering that neural networks are increasingly being used in safety-critical scenarios such as autonomous driving [38] and medical diagnostics [4]. With the increasing popularization and use of QNNs, it is urgent to develop efficient and effective analysis methods that also provide rigorous guarantees. Formal verification is an established technique which applies mathematical reasoning to provide such guarantees. In this paper, we explore the use of formal methods to quantitatively analyze the changes in a neural network's output for the same input space before and after quantization.

We introduce a generalized concept of equivalence to characterize the differences between an original network and a quantized network. Specifically, we say that a real-valued DNN $\mathcal{N}(x)$ and its quantized version $\mathcal{N}_Q(x)$ are δ-equivalent if, for all possible inputs x in an input space B of interest, $\|\mathcal{N}(x) - \mathcal{N}_Q(x)\|_\infty < \delta$ holds. We refer to it as "equivalence" because the goal is for the two networks to produce outputs that are as similar as possible for every possible input. Of course, it is unrealistic to expect a QNN to maintain absolute equivalence with its original real-valued network. Therefore, a tolerance δ is introduced.

Our approach for determining δ-equivalence is inspired by a stream of related work on *error bound verification*. Paulsen et al. [29, 30] propose differential verification methods, which aim to establish formal guarantees on the difference between a network represented by weights in high-precision floating-point format and another network represented by weights in low-precision fixed-point format. The low-precision network can be thought of as a simplified and partially quantized neural network. They apply a sound but incomplete interval propagation to overestimate the difference between two networks. Zhang et al. [41] extend the differential analysis with a complete mixed integer linear programming (MILP)-based method in order to precisely verify error bounds between real-valued neural networks and fully quantized networks obtained using a toy quantization scheme. Their analysis requires formally modeling both networks, meaning that the scale of the problem is double that of formally analyzing one of the networks. Even for small networks, differential analysis using this approach often cannot be completed within a reasonable amount of time. As a way to simplify the problem, Zhang et al. [41] focus on classification networks and only on the output difference for the predicted class rather than all classes. Besides this simplification, the toy quantization scheme used in their work is simpler than and different from quantization schemes used in practice. As a result, conclusions

drawn from their approach may not be valid for quantized networks obtained with standard quantization packages.

In this work, we build on and extend these approaches in several ways. We follow Zhang et al. by using two neural networks and MILP encodings and solvers, which are exact in the sense that they guarantee both *soundness* and *completeness*. However, in contrast to the toy quantization schemes used in Zhang et al., we focus on realistic quantization schemes used in popular deep learning frameworks such as PyTorch and TensorFlow. Furthermore, to overcome the scalability challenges posed by using exact techniques and two networks, we propose an efficient parallel verification method. Our main framework is based on a divide-and-conquer strategy, which breaks down the original problem into many sub-problems and uses multiple CPUs to solve them simultaneously. This approach has been shown to be effective for related problems such as Boolean satisfiability [15, 16].

We observe two main challenges when using divide-and-conquer. First, it is difficult to divide the problem into sub-problems of similar difficulty. Some sub-problems can be extremely simple, to the extent that the overhead of divide-and-conquer is larger than the runtime of the sub-problem. Conversely, other sub-problems are almost as challenging as the original problem, requiring nearly the same amount of time to solve. Second, even though sub-problems are always theoretically simpler than the original problem, in practice, sub-problems can take longer to solve due to the high variance in runtime performance under small perturbations, a feature that is common among solvers that tackle NP hard (or harder) problems. To address the first issue, we propose a two-stage output-based partition method. The first stage eliminates logical disjunction from the equivalence property, and the second stage further decomposes the problem based on estimated interval ranges for the output nodes provided by standard abstract interpretation-based analyses. To address the second issue, we adopt a two-level parallel strategy. We utilize multiple independent processes to solve all sub-problems concurrently. Within each process, we employ multi-threaded parallelism to run a portfolio of solving algorithms, each with a different configuration, all solving the same sub-problem simultaneously. Our parallel algorithm is designed for high performance computing (HPC) clusters which can utilize hundreds of CPUs. To minimize the additional overhead caused by process communication, inter-node communication uses the standard message passing interface (MPI), while multi-core parallelization within processes is achieved through multiprocessing and shared memory mechanisms.

We call our system **E**fficient **Q**NN **E**quivalence **V**erification, or EQEV. Our experimental results show that EQEV can, for the first time, verify the equivalence of QNNs of moderate size in non-trivial scenarios. Our contributions can be summarized as follows:

1. We provide the first MILP-based exact equivalence verification approach for QNNs obtained with quantization schemes used in current popular deep learning frameworks;

2. We propose an output-based two-stage partitioning method which helps balance the difficulty of sub-problems and greatly improves the efficiency of parallel verification;
3. We implement our approach in a tool called EQEV that can utilize hundreds of CPUs in an HPC cluster for equivalence verification, achieved through a combination of MPI, multi-processing, and multi-threading. EQEV is the first system to verify the equivalence of moderate-sized networks in reasonable time.

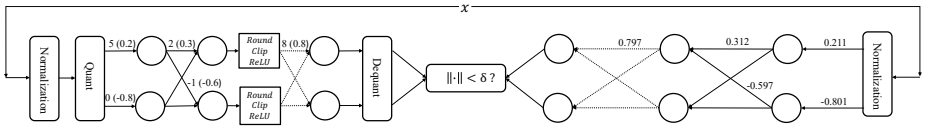

Fig. 1. δ-equivalence of quantized networks and real-valued networks.

2 Related Work

Formal verification of DNNs is a technique which determines whether a network satisfies a formal property by either proving the property or providing a counter-example to the property. For instance, robustness is often formulated as a formal property expressing the non-existence of adversarial examples in a bounded neighborhood of some input based on a certain distance metric. Researchers have developed various techniques for verifying DNNs such as constraint solving (based on satisfiability modulo theories (SMT) [23, 8, 20, 37] or MILP [3, 9, 7]) and abstract interpretation [36, 35, 10, 40, 31, 27, 33]. The former provides sound and complete guarantees but with limited scalability as the problem is NP-hard [23]; the latter utilizes over-approximation to improve scalability but at the cost of completeness. However, work on formal approaches for QNN verification is limited, potentially due to the difficulty of modeling quantization schemes or the computational cost [22].

Not until recently did researchers start to focus on the verification of QNNs. For example, Henzinger et al. [14], Mistry et al. [28], and Zhang et al. [42] propose using SMT, MILP, and integer linear programming (ILP), respectively, to model the QNN verification problem. Although this work pioneers new directions for QNN verification, it is based on toy quantization schemes rather than realistic quantization algorithms used in popular deep learning frameworks such as PyTorch and TensorFlow. Very recently, in 2023, Huang et al. [19] provide a hybrid verification method that is the first to support realistic quantization schemes in PyTorch.

Nevertheless, the above-mentioned DNN and QNN verification techniques only focus on a single individual neural network; they are inadequate when

performing formal analyses involving two networks. Paulsen et al. [29, 30] propose a differential verification method leveraging abstract interpretation to verify the discrepancy between two neural networks with identical topologies but slightly different weights. Zhang et al. [41] build on this to precisely verify the error bound between real-valued and quantized models using a complete MILP analysis, but still only for toy quantization schemes and only for a predicted class (as opposed to all classes) in a classification task.

3 Preliminaries

3.1 Quantized Neural Networks (QNNs)

Quantization, in our context, is the process of mapping a variable from a real number r to an integer q using parameters s and z, as follows.

$$\text{Quantization: } q = Round(\frac{r}{s} + z), \text{ De-quantization: } r = s(q - z). \quad (1)$$

We call Equation (1) the *quantization scheme*, and constants s and z the quantization parameters, for which s ("scale") is an arbitrary real number, and z ("zero point") is the integer corresponding to q when $r = 0$. In practice, the quantized value q is represented by a fixed number of bits, e.g., q is an 8-bit integer in 8-bit quantization. If q does not fit in the number of bits provided, then the closest representable value is used.

Matrix multiplication is one of the most common operations in DNN forward inference. Assume there are three matrices of real numbers, and the product of the first two matrices is the third matrix. We use $r_\alpha^{(i,j)}$, with $\alpha \in \{1, 2, 3\}$ and $0 \leq i, j \leq N - 1$, to denote the entries of these three matrices, assuming they are all square and of size N. In order to be able to handle neurons with distinct quantization parameters, we use (s_α, z_α) to denote the quantization parameters for the three matrices and $q_\alpha^{(i,j)}$ to represent the quantized entries. Using the quantization scheme in Equation (1), we have $r_\alpha^{(i,j)} = s_\alpha(q_\alpha^{(i,j)} - z_\alpha)$. Thus, by 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

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

For a dense layer of a DNN, suppose we use $\mathbf{y} := ReLU(W\mathbf{x} + \mathbf{b})$ to denote the nonlinear transformation, where W is the weight matrix, $\mathbf{b}$ is the bias vector, $\mathbf{x}$ is the input vector, and $\mathbf{y}$ is the output vector. Then, its quantized version can be written as $\mathbf{y}_q := g(\mathbf{x}_q, W_q, \mathbf{b}_q)$, where W_q, $\mathbf{b}_q$, $\mathbf{x}_q$, and $\mathbf{y}_q$ are the

quantized entries of W, $\mathbf{b}$, $\mathbf{x}$, and $\mathbf{y}$, respectively, and g is explained below. For the input and output vectors, we use $z_{\mathbf{x}}$, $z_{\mathbf{y}}$ as their zero points and $s_{\mathbf{x}}$, $s_{\mathbf{y}}$ as their scales. And we use z_w^j to denote the zero point and s_w^j to denote the scale for the weight matrix corresponding to the j-th output neuron. Following [19], the computation $\mathbf{y}_q := g(\mathbf{x}_q, W_q, \mathbf{b}_q)$ can be written as the series of calculations shown in Equation (4).

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

Here, the *Clip* function returns a value for $\hat{y}_2^j$ in the range $[lb, ub]$ that is closest to $\hat{y}_1^j$. The values for lb and ub are chosen based on the smallest and largest values allowed by a the quantization scheme, e.g., $[lb, ub] = [0, 255]$ for an 8-bit unsigned type. Huang et al. [19] further propose an ILP encoding for such quantization schemes. We review the scheme and flag variables that are turned into ILP variables by adding a dot above them, e.g. $\dot{x}_q^i$ for the input. Step (i) becomes

$$\hat{y}_0^j = z_y + \frac{s_w^j s_x}{s_y} \sum_i (w_q^{(i,j)} - z_w^j)(\dot{x}_q^i - z_x) + b_q^j. \tag{5}$$

We do not introduce an ILP variable for $\hat{y}_0^j$ as we will be able to eliminate it below. The *Round* function in Step (ii) can be encoded by the following two constraints.

$$\begin{cases} \dot{\hat{y}}_1^j - \hat{y}_0^j \leq 0.5 \\ \hat{y}_0^j - \dot{\hat{y}}_1^j \leq 0.5 - \varepsilon, \end{cases} \tag{6}$$

for some small value of ε. Eliminating the temporary variable $\hat{y}_0^j$ by combining (5) and (6), we have

$$\begin{cases} \dot{\hat{y}}_1^j - z_y - \frac{s_w^j s_x}{s_y} \sum_i (w_q^{(i,j)} - z_w^j)(\dot{x}_q^i - z_x) - b_q^j \leq 0.5 \\ z_y + \frac{s_w^j s_x}{s_y} \sum_i (w_q^{(i,j)} - z_w^j)(\dot{x}_q^i - z_x) + b_q^j - \dot{\hat{y}}_1^j \leq 0.5 - \varepsilon. \end{cases} \tag{7}$$

The *Clip* function is encoded as

$$\mathit{Encode\_max}(\dot{\hat{y}}_{max}^j, \dot{\hat{y}}_1^j, lb) \bigcup \mathit{Encode\_min}(\dot{\hat{y}}_2^j, \dot{\hat{y}}_{max}^j, ub), \tag{8}$$

where $\dot{\hat{y}}_{max}^j$ denotes a fresh auxiliary variable. $\mathit{Encode\_max}()$ and $\mathit{Encode\_min}()$ can both be realized in a linear form with the big-M method [3, 19]. Finally, Step (iv) takes the maximum of $\hat{y}_2^j$ and the zero point z_y, which represents the $ReLU$ function in the quantized network. This can be directly encoded as $\mathit{Encode\_max}(\dot{\hat{y}}_q^j, \dot{\hat{y}}_2^j, z_y)$.

We note that the quantization scheme in PyTorch usually represents inputs and outputs of each layer as unsigned integers and weights as signed integers. Also, quantization parameters (i.e., zero points and scales) are determined at the time of quantization; therefore they are constants at inference time. Additional encodings for typical fusion layers such as the fusion of affine transformations and batch normalizations, as well as the fusion of affine transformations and *ReLUs*, are detailed in [19], and we will not review them here. For real-valued networks, the MILP encoding for each layer of the computation process $\mathbf{y} := ReLU(W\mathbf{x} + \mathbf{b})$ is well-established (see, e.g., [3, 9, 7]).

4 δ-Equivalence

As mentioned above, quantized neural networks can exhibit behavior that is different from the original network. These differences are primarily due to the precision loss introduced by parameter, input, and output quantization. During the quantization process, floating-point numbers are typically converted into fixed-bit representations with lower precision (e.g., 8-bit integers). This means that the parameter values from the original network may be truncated or rounded, introducing quantization errors. These errors can accumulate during the forward propagation of the network, resulting in discrepancies between the output values of the quantized and the original networks. Such discrepancies can negatively impact a network's performance, especially for tasks that require high-precision outputs. Thus, a critical goal in neural network quantization is to minimize the impact of the quantization process on accuracy and generalization. A natural way to approach this is to aim to minimize the differences in the output values between the QNN and the original network. By applying formal techniques, we can provide rigorous guarantees about the impact of quantization, which can then be used to inform the quantization process and reduce errors. This paper reports on our efforts to do precisely this.

We denote a real-valued DNN with d layers as $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$, which can be seen as a composition of a set of d functions such that $\mathcal{N} := \bar{l}^d \circ \bar{l}^{d-1} \circ ... \circ \bar{l}^1$. A quantized neural network $\mathcal{N}_Q : \mathbb{R}^n \to \mathbb{R}^m$ with d layers can be seen as a composition of $d+2$ functions such that $\mathcal{N}_Q := l^D \circ l^d \circ l^{d-1} \circ ... \circ l^1 \circ l^Q$ where $l^Q : \mathbb{R}^n \to \mathbb{Z}^n$ is a quantization layer for the inputs and $l^D : \mathbb{Z}^m \to \mathbb{R}^m$ is a de-quantization layer producing the outputs. Intermediate layers $l^d \circ l^{d-1} \circ ... \circ l^1$ are maps from $\mathbb{Z}^n$ to $\mathbb{Z}^m$.

Due to the use of real arithmetic, it is difficult (if not impossible) to achieve exact numerical equivalence between two different neural networks. Thus, for our purposes, if the output difference between the quantized and the original networks is less than a given constant δ, we say they are equivalent. Intuitively, smaller values of δ imply stronger notions of equivalence, also indicating that the quantized network better preserves the original functionality.

We will use the following definition of local equivalence between a real-valued neural network and its quantized counterpart, which focuses on the differences in output values for input regions around the data distribution. Our definition

is similar to those found in [29, 41], except that we focus on differences over an entire output vector rather than differences only over a specific target output.

Definition 1 (Local δ-Equivalence). *Given a DNN $\mathcal{N}$ and its quantized neural network $\mathcal{N}_Q$, they are considered to be* ***locally δ-equivalent*** *at point x_* with respect to a radius r if the following formula holds:*

$$\forall x.\ (x \in B_\infty(x_*, r) \rightarrow \|\mathcal{N}(x) - \mathcal{N}_Q(x)\|_\infty < \delta), \tag{9}$$

where $B_\infty(x_*, r) = \{x \mid \|x - x_*\|_\infty \leq r\}$ denotes the input neighborhood around x_*, and δ is some non-negative constant.

Then we extend the concept of local δ-equivalence to global δ-equivalence, which captures the differences caused by quantization across the entire possible input space (even though this space may contain meaningless input examples). It can be used to measure the overall difference in performance between two models. As before, a smaller δ implies that the quantized model preserves better the performance of the original real-valued model.

Definition 2 (Global δ-Equivalence). *In Definition 1, if the perturbation radius r is large enough so that the input neighborhood $B_\infty(x_*, r)$ covers all possible inputs, the DNN $\mathcal{N}$ and QNN $\mathcal{N}_Q$ are said to be* ***globally δ-equivalent****.*

For example, in the case of image inputs where each pixel value is in the range of $[0, 255]$, if Definition 1 holds when r is set to 255, then we say that the original and the quantized models have global δ-equivalence.

5 MILP Encoding

We next discuss how to encode Equation (9) as a MILP problem. Note that, if we only need to check a property on a single quantized network, we can model the input as the value directly after preprocessing and thus skip the encoding of the quantization layer. However, when it comes to verifying δ-equivalence, we cannot ignore the quantization layer. This is because, for the common input shared by the original and the quantized models, the quantized model first sends the input through the quantization layer while the original model does not. We use $\dot{x}_\gamma^j$ to denote the (real-type) variable representing the j-th neuron of the real-valued input, and $\dot{x}_q^j$ to denote the (integer-type) variable representing the value quantization. For the quantization layer l^Q, we have:

$$\begin{cases} \dot{x}_\gamma^j / s_x + z_x - \dot{x}_q^j \leq 0.5 \\ \dot{x}_q^j - (\dot{x}_\gamma^j / s_x + z_x) \leq 0.5 - \varepsilon. \end{cases} \tag{10}$$

Similarly, for the output layers of the two networks, we use real-type variables $\dot{\mathcal{N}}_Q^j, \dot{\mathcal{N}}^j$ to denote the j-th neuron of the outputs of l^D (QNN) and ℓ^d (DNN), respectively. The integer-type variable $\dot{y}_q^j$ represents the j-th neuron of the output of quantized layer l^d. We have

$$\dot{\mathcal{N}}_Q^j = s_y(\dot{y}_q^j - z_y), \tag{11}$$

where z_y and s_y are the zero point and the scale of the de-quantization layer.

To examine whether the equivalence property holds, we negate Formula (9) and check for satisfiability. We rewrite the constraint $\|\mathcal{N}(x) - \mathcal{N}_Q(x)\|_\infty < \delta$ as

$$Encode\_Max(\dot{y}_\delta, |\dot{\mathcal{N}}_Q^1 - \dot{\mathcal{N}}^1|, |\dot{\mathcal{N}}_Q^2 - \dot{\mathcal{N}}^2|, \ldots, |\dot{\mathcal{N}}_Q^m - \dot{\mathcal{N}}^m|) \wedge (\dot{y}_\delta \geq \delta), \quad (12)$$

where $\dot{y}_\delta$ represents the maximum output difference. We have $|x| = max(-x, x)$, so Formula (12) can be transformed into a linear form. Formulae (10), (11), and (12) jointly model the DNN and the QNN, while their respective layer-by-layer computation processes can be represented using the encoding described in Section 3.

6 Symbolic Interval Analysis

In this section, we introduce a process for conducting interval analysis on two networks simultaneously in order to obtain bounds for each variable. After obtaining the bounds, it is sometimes possible to directly conclude that the property holds. This is the approach taken in [29, 30]. However, even if the bounds are not precise enough to prove the property, we can use them to simplify the MILP problem. In particular, the bounds may be able to reduce the search space and show that some neurons are always active or always inactive. Finally, the bounds are also useful when deciding how to divide the problem in the parallel approach we describe next. To compute the bounds, we use standard abstract interpretation techniques which use convex polyhedra to over-estimate the output interval of each node [34].

In order to apply these standard techniques, we first combine our two networks into a single network by adding expressions of the form:

$$y_\delta^j := \mathcal{N}_Q^j - \mathcal{N}^j \quad (13)$$

In other words, we add one more layer with weights consisting solely of 1 and -1 in order to calculate the symbolic difference between the outputs of the two networks. We can now regard the combined networks as a single network. As before we use a symbol with a dot above it to represent a variable in the MILP model and use existing techniques to compute the lower bound $lb(\dot{y})$ and upper bound $ub(\dot{y})$ for each variable $\dot{y}$. After obtaining the bounds of variable $\dot{y}_\delta^j$ in particular (the variable corresponding to y_δ^j), sometimes we can directly conclude that the property holds. When verifying equivalence, bound propagation can sometimes also allow us to conclude that the property does not hold. The possible results are summarized as follows.

1. *If there exists a* $j \in \{1, 2, ..., m\}$ *such that* $lb(\dot{y}_\delta^j) \geq \delta \vee ub(\dot{y}_\delta^j) \leq -\delta$, *then the* δ*-equivalence property* ***does not hold****.*
2. *If, for all* $j \in \{1, 2, ..., m\}$, *we have* $lb(\dot{y}_\delta^j) > -\delta \wedge ub(\dot{y}_\delta^j) < \delta$, *then the* δ*-equivalence property* ***holds****.*
3. *Otherwise, it is* ***unknown*** *whether the* δ*-equivalence property holds.*

7 Parallelization

As the verification radius (r in Equation (9)) increases and the value of delta approaches the maximum value where Equation (9) still holds, the interval analysis method will eventually fail. Moreover, the participation of two networks doubles the size of the verification problem. This means that for many radii and tolerances δ of interest, existing techniques are unable to obtain any results within a reasonable time using the computing power of a single machine. Therefore, in this section, we introduce a distributed parallel method to tackle such cases.

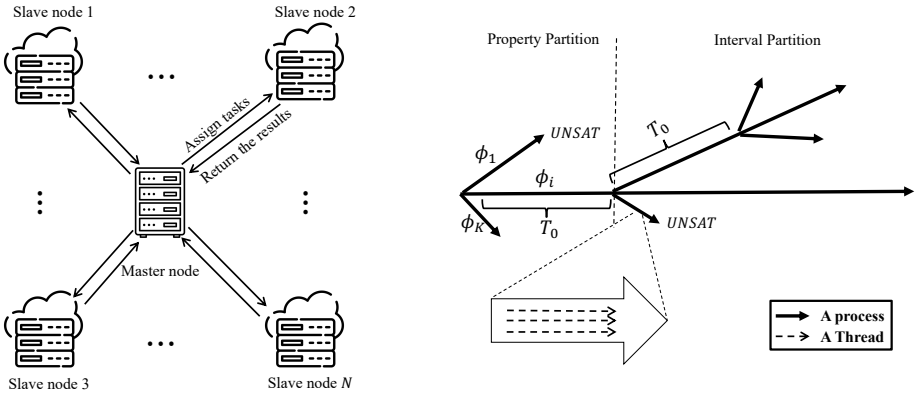

Fig. 2. The parallel verification framework.

7.1 Process Management

Divide-and-conquer has been shown to be an efficient method for conquering difficult constraint solving problems and has even been used to solve combinatorial mathematical puzzles such as the Pythagorean triples problem[15–18]. Wu et al.[37] showed that divide-and-conquer can significantly improve the efficiency of verifying real-valued DNNs. For these reasons, we also adopt a divide-and-conquer paradigm. However, unlike previous work, we adopt two different levels of parallelism and improve the process management strategy as explained below.

We partition a problem (a formula ϕ) into K independent sub-problems, ϕ_i, where $\phi \leftrightarrow \vee_{i\in\{K\}}\phi_i$, and try to solve each sub-problem within a time budget T_0. If a solving attempt exceeds the time budget, that problem is further partitioned and the sub-problems are allocated the same time budget. Our parallel problem-solving approach can be regarded as a tree as shown in Fig. 2. Assuming some number N of CPUs, the problem can be partitioned into no more than N sub-problems initially. We wait for a duration of T_0 before continuing to partition sub-problems because, as the partition level increases, the number of sub-problems grows exponentially. After time T_0 has elapsed, some (simpler) sub-problems

may be finshed, freeing up computing resources. The remaining sub-problems are further partitioned. Unlike in previous work, we do not halt the process of solving a problem when further splitting it into sub-problems. We observed that doing so often results in situations where problems, having already consumed considerable time and nearing been solved, are halted. Consequently, we end up expending a substantial amount of time to re-solve the same problem in the form of many sub-problems.

Utilizing CPUs on a single machine is insufficient for solving hard equivalence verification problems. Thus, to enable the use of hundreds of CPUs in an HPC cluster, we utilize the MPI protocol to facilitate communication among computing nodes. To reduce the additional overhead introduced by process communication, we mix this multi-node approach with a shared memory multi-process model for utilizing multiple cores within a single computing node. We adopt the classical master-slave model to manage processes. In the master process, we maintain a tree structure to record the status of each sub-problem solving process. If the master node sends the complete formula to the slave nodes every time, the communication overhead would be significant. Therefore, we initialize a basic formula for each node, and then each time the master node distributes tasks, it only sends an incremental modification of the formula to the slave nodes.

When solving of hard instances, there is a very interesting phenomenon whereby sub-problems might take more time to solve than the original problem. This is because, in order to solve problems that are NP-hard, modern solvers use a large number of heuristic strategies, with the result that solving times are highly unstable: small changes to a problem can result in highly variant runtimes (both faster and slower). To alleviate this issue, we implement a second level of parallelism within each process that is solving each sub-problem. Namely, within each process, we use multiple threads to run solving algorithms with different parameter configurations to simultaneously process the same sub-problem. Compared to the divide-and-conquer approach, the advantage of a parallel paradigm using solving algorithms with multiple different parameter configurations to handle the same problem is that it can simultaneously explore different search branches of the problem without implicitly increasing the difficulty of the problem. The divide-and-conquer method, while decomposing the original problem into several sub-problems, shifts the problem from finding a single solution to finding multiple different solutions which implicitly increases the difficulty of the problem. Our hybrid parallel approach not only makes full use of computing resources but also helps ensure more stable performance in solving sub-problems.

7.2 Partitioning Strategies

We now discuss our partitioning strategy for equivalence verification problems. When using the divide-and-conquer approach to handle NP-hard problems, it is common to encounter a situation where, e.g., after dividing a problem into 10 sub-problems, most of them (e.g., 9) are very simple and only take a few seconds to solve, while one sub-problem is as hard as the original problem. This

is because sometimes the sub-problems we generate actually correspond to very trivial situations, or in other words, these situations correspond to search spaces that can be avoided by the reasoning algorithm after a few reasoning steps. To generate more balanced sub-problems, Wu et al. [37] propose a $ReLU$-based partitioning which creates case splits that fix the phase of $ReLUs$ directly. For real-valued neural networks, the more $ReLU$ function states that are fixed, the simpler the sub-problems become. In the extreme case, if all $ReLUs$ states are fixed, a linear programming problem is obtained that can be efficiently checked. However, for QNN verification, this is not the case, as even if we fix the states of all the $ReLU$ functions, the verification problem is still an NP-hard MILP or ILP problem. Moreover, we found empirically that this decomposition method is not very effective for the verification of QNNs.

We propose a two-stage output-based partitioning method for QNN verification with the goal of producing sub-problems of (hopefully) equal difficulty. The first stage (property partitioning) is to partition the search space by eliminating logical disjunctions ($\vee$) from the property being checked. The second stage (interval partitioning) is to decompose by doing case analysis over the range of the variable corresponding to the output.

Property Partitioning In our property partition stage, formula (12) can be be rewritten in the following form:

$$\underbrace{(\dot{\mathcal{N}}_Q^1 - \dot{\mathcal{N}}^1 \geq \delta)}_{1} \vee \underbrace{(\dot{\mathcal{N}}^1 - \dot{\mathcal{N}}_Q^1 \geq \delta)}_{2} \vee, ..., \vee \underbrace{(\dot{\mathcal{N}}_Q^m - \dot{\mathcal{N}}^m \geq \delta)}_{2m-1} \vee \underbrace{(\dot{\mathcal{N}}^m - \dot{\mathcal{N}}_Q^m \geq \delta)}_{2m} \tag{14}$$

The $2m$ items connected via disjunction in the formula (14) correspond to $2m$ sub-problems. Disjunctions can often hinder the efficiency of MILP algorithms; thus, this decomposition can be very helpful for improving the efficiency of reasoning.

Interval Partitioning In the interval partitioning stage, we further partition the problem based on the computed interval range of variable $\dot{\mathcal{N}}_Q^j$, which corresponds to the output of the j-th component of the quantized neural network. Note that this interval consists of integers. Suppose we have a sub-problem ϕ' containing $\dot{\mathcal{N}}_Q^j$, whose current interval is $[lb, ub]$. We can partition ϕ into K sub-problems, where the i-th sub-problem ($1 \leq i \leq K$) is:

$$\phi_i' := \phi' \wedge lb + \lfloor \frac{ub - lb}{K} \rfloor (i-1) \leq \dot{\mathcal{N}}_Q^j \leq \min(lb + \lfloor \frac{ub - lb}{K} \rfloor i - 1, ub) \tag{15}$$

Here we choose to decompose the output variable of the QNN instead of decomposing the output variable of the real-valued network for two reasons: (1) The interval of the QNN output variable consists of integers, and the smallest size it can be decomposed into is 1, whereas a real number interval can be infinitely subdivided. (2) After the first stage of decomposition, variables $\dot{\mathcal{N}}_Q^j$ and

$\dot{\mathcal{N}}^j$ only appear together in a single constraint, in the form of $(\dot{\mathcal{N}}_Q^j - \dot{\mathcal{N}}^j \geq \delta)$ which is a binary constraint. As soon as we modify the interval of variable $\dot{\mathcal{N}}_Q^j$, the reasoning algorithm will adjust the value range of variable $\dot{\mathcal{N}}^j$ through a simple reasoning step. Moreover, dividing the output space of the network into very small regions indirectly constrains the input space and the states of many activation functions.

Advantages In terms of the number of sub-problems generated, the method based on $ReLU$ function state decomposition produces 2^n sub-problems if n $ReLUs$ are decomposed at a time. In equivalence verification, two networks are involved, doubling the number of $ReLUs$. Sometimes, even after repeatedly executing decomposition to fix the states of dozens of neurons, the effect on the entire network may not be very significant. In our method, the problem decomposition in the first stage only produces a number of sub-problems that is twice the number of neural network output categories. For many neural networks, the number of output categories is not too large. Thus, we can typically obtain enough CPUs to process the sub-problems generated in the first stage. After running these with a time budget of T_0 and filtering out the solved cases, we are ready to enter the second stage of partitioning. In the second stage, we limit the number of sub-problems, K, to be less than the number of available CPUs on a node in order to fully utilize the computational power of a node and minimize communication among nodes as much as possible. For many moderate verification radii, the interval sizes obtained for QNN output variables are often even smaller than the number of CPUs in a single node.

With regards to balancing problem difficulty, for the $ReLU$ function state decomposition method, since the activation states of the $ReLU$ function are estimated based on an over-approximation method, there will be some sub-problems for which activation states do not exist. These situations are likely trivial, but we still need to spend a lot of time running and filtering them, while other activation states correspond to situations with the same difficulty as the original problem. In contrast, we know that each output node of the neural network corresponds to a function almost the size of the network itself. Thus, in our method, by decomposing the problem based on the output, although there may be some non-existent cases in the interval estimated by the over-approximation method, for the possible output values, the difficulty of each sub-problem is fairly balanced.

8 Experiments

We implemented a Python-based parallel verification tool called EQEV.[6] We use Gurobi [12] as our backend MILP solver and the abstract interpretation-based interval analysis is done by Marabou [24]. Our experiment is conducted on two well-known neural network architectures: fully connected neural networks

[6] https://github.com/huangdiudiu/EQEV

(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 FC2-100 is 784 $\times$ 100$\times$ 100 $\times$ 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 [26] and quantized with PyTorch using its default static quantization scheme. Verification experiments are conducted on the test set. The experimental environment is a high-performance computing (HPC) cluster running slurm for cluster management. For EQEV, unless specifically stated otherwise, the default configuration uses 21 compute nodes with 16 CPUs each, for a total of 336 CPUs. We allocate 1GB of memory for each CPU. The parameters of the CPU are not fixed because the HPC is heterogeneous, containing several different types of CPUs. During the interval decomposition stage, we set the value of K to 14.

We verify the local δ-equivalence of networks with $r = 4, 8, 12$ and $\delta = 1.7, 1.9, 2.1, 2.3$. For the quantized networks we use, choosing these values for δ makes the problems relatively difficult. For most instances, the verification results switch from SAT to UNSAT near these values of δ. In other words, the values of δ are situated near the phase transition points (the maximum difference between two networks). For the QNNs we use, no instances can be directly solved by interval analysis. We randomly select 100 examples from the test set, and any instance that exceeds the timeout is recorded as 1800 seconds (30min).

It should be emphasized that with 100 test cases for each setting of δ and r, and considering we have 4 different networks and 4 types of solving algorithms, with each instance requiring 30 minutes of computation time and over 300 CPUs, conducting a fully exhaustive set of experiments is challenging. Thus, we selected a subset of representative scenarios for each comparison.

9 Efficiency

We first compare the parallel methods that come with Gurobi and the parallel method of ReLU-based partitioning. The basic parallel approach of Gurobi is to run multiple solvers with different parameter configurations on the same problem. Since many nodes in our cluster have a maximum of only 20 CPUs, we set it up to use 20 threads, and we name it "20_Threads". We name the parallel approach of ReLU-based partitioning "relu_split." Since the original version does not support QNNs nor our cluster environment, we replicated a similar version using MPI based on the literature [37]. It utilizes the same number of CPUs as EQEV.

Table 1 demonstrates that EQEV outperforms other parallel methods for quantization equivalence verification. The table shows total runtimes for each configuration with the number of timeouts in parentheses. We used a timeout of 1800 seconds. The results of 20_Threads indicate that without decomposing

Table 1. Total execution time(s) of different methods.

			$\delta = 1.7$	$\delta = 1.9$	$\delta = 2.1$	$\delta = 2.3$
FC1-100	$r = 4$	20_Threads	177677(97)	158135(86)	91210(46)	39125(18)
		Relu_Split	180000(100)	162468(85)	92314(46)	35505(16)
		EQEV	**130860(36)**	**128242(61)**	**52502(25)**	**17945(6)**
	$r = 8$	20_Threads	179986(99)	178721(99)	176795(98)	170379(93)
		Relu_Split	179958(97)	178252(99)	180000(100)	180000(100)
		EQEV	**127016(27)**	**165912(82)**	**157961(79)**	**148479(73)**
	$r = 12$	20_Threads	180000 (100)	180000(100)	180000(100)	180000(100)
		Relu_Split	1751472(97)	180000(100)	180000(100)	180000(100)
		EQEV	**131845(27)**	**175960(93)**	**179334(98)**	180000(100)
FC2-100	$r = 4$	20_Threads	178524(99)	180000(100)	180000(100)	180000(100)
		Relu_Split	170304(92)	177566(98)	178838(99)	180000(100)
		EQEV	**124919(35)**	**153233(62)**	**163313(73)**	**174654(92)**
	$r = 8$	20_Threads	177135(98)	180000(100)	180000(100)	180000(100)
		Relu_Split	1570525(86)	180000(100)	180000(100)	180000(100)
		EQEV	**102884(23)**	**144899(52)**	**173289(81)**	**176651(91)**
	$r = 12$	20_Threads	166140(90)	178470(99)	178359(99)	180000(100)
		Relu_Split	1571382(87)	178769(99)	178314(99)	180000(100)
		EQEV	**98470(23)**	**145065(52)**	**173184(88)**	**178002(97)**

the problem, merely using different parameter configurations to explore various branches simultaneously is almost completely ineffective for these difficult instances. Relu_Split also performs poorly here for at least two reasons: first, it partitions the problem and then restarts after running for a while, so a lot of time is wasted resolving problems that time out; second, since equivalence verification involves two networks, the number of ReLU nodes doubles. During its execution, we found that sub-problems of the neurons initially chosen for decomposition cannot be solved within the time budget. After decomposing only 9 ReLU nodes, the number of the sub-problems ($2^9 = 512$) exceeds the number of CPUs (336), causing process blocking. Our method decomposes the problems into fewer partitions with relatively more balanced difficulty. Moreover, our process management approach is better suited for verifying equivalence. We do not stop the processes solving the original problem (the problem before interval partitioning); sometimes, this original problem is able to produce a result shortly after the timeout budget, which means we do not have to spend substantial time on restarting. Additionally, in our process management strategy, the MPI+multi-process design pattern reduces the overhead of process communication, while the multi-threading parallelism within a process enhances the efficiency and stability of solving individual sub-problems.

We also include a rough comparison with the verifier of Zhang et al. [41]. We modify their code to support verifying the equivalence defined in our paper and set up 20 parallel threads for its back-end Gurobi solver. The structure of network FC2_100 used for comparison is the same, but the network parameters and

quantization scheme are different. The normalization applied to inputs is also different, depending on how the neural network is trained. Our network inputs are normalized to the range of $[-1, 1]$ using standard normalization, whereas Zhang et al.'s network inputs are normalized to $[0, 1]$ by dividing by 255. This means that the same values of δ that are difficult for our verifier may not necessarily be difficult for their verifier.

Table 2. The comparison between QEBVerif and EQEV on FC2-100.

		$r = 4$		$r = 8$		$r = 12$	
		QEBVerif	EQEV	QEBVerif	EQEV	QEBVerif	EQEV
$\delta = 1.7$	Time(s)	154642	124919	158843	102884	162509	98470
	Unknown	85	35	86	23	88	23
	Abstract	6	0	0	0	0	0
$\delta = 1.9$	Time(s)	143622	153233	163460	144899	162582	145065
	Unknown	79	62	90	52	89	52
	Abstract	13	0	0	0	0	0
$\delta = 2.1$	Time(s)	130850	163313	166283	173289	169864	173184
	Unknown	71	73	91	81	93	88
	Abstract	20	0	0	0	0	0
$\delta = 2.3$	Time(s)	114479	174654	169744	176651	169492	178002
	Unknown	62	92	94	91	93	97
	Abstract	30	0	0	0	0	0

Table 2 presents the experimental results, from which we can observe that EQEV is capable of verifying more instances within the time limit. The instances solved during the interval analysis phase are listed in the row of "Abstract." As the δ values are not near the phase transition points for the networks used by QEBVerif, we find that almost all instances solvable by QEBVerif in our experiments are those that can be directly resolved by interval analysis within 5 seconds, while hardly any of the instances requiring MILP solving can be solved within the time limit. However, for our verification, the setting of δ is such that no instances can be directly resolved through interval analysis and all results are provided by the MILP solving phase.

Table 1 and Table 2 list some of the more difficult cases. For some relatively simpler cases, our method can almost verify all instances. For example:

- For FC1-100, when $\delta = 1$, $r = 4, 8, 12$, EQEV can verify all 100 instances within a total time of 4000s, and the results are all SAT. When $delta = 3$, $r = 4$, EQEV can verify all 100 instances within a total time of 2633s, and the results are all UNSAT.
- For FC2-100, when $\delta = 1$, $r = 4$, EQEV can verify all 100 instances within a total time of 17955s, and the results are all SAT. When $\delta = 1$, $r = 12$, EQEV can verify all 100 instances within a total time of 11715s, and the results are all SAT.

– For CNN1, when $\delta = 1$, $r = 4, 8, 12$, EQEV can verify all 100 instances within a total time of 2000s, and the results are all SAT. When $\delta = 3$, $r = 4$, EQEV can verify all 100 instances within a total time of 1432s, and the results are all UNSAT.

10 Verification Results

10.1 Local δ-Equivalence

Fig. 3 plots the percentage (number) of instances shown to be UNSAT (equivalent) by EQEV at each tolerance. The curve of "UNSAT+Unknown" can be regarded as the upper bound of the percentage of instances that have equivalence. As the value of δ increases, it is clear that the curve corresponding to "UNSAT" must first diverge from and then gradually converge with the curve corresponding to "UNSAT+Unknown". The point of maximum divergence corresponds to the most difficult scenario in our equivalence verification. Theoretically, two curves will intersect at points where the value of δ is very small and where the value of δ is very large.

10.2 Global δ-Equivalence

For global δ-equivalence, we set r large enough to cover the entire input space and add cases with $\delta = 5, 15, 20$. Table 3 presents the results of global equivalence verification. We can observe that for these δ values, almost no network is able to maintain good equivalence throughout the entire input space, not even for high values of δ. This indicates that the step of adjusting scale and zero point with the training set during the quantization process effectively preserves the equivalence of the quantized neural network and the original real-valued neural network in the data distribution space only. The regions that are unsafe for $\delta = 15, 20$ may exhibit significant differences between the QNN and its original real-valued NN.

Table 3. Global δ-Equivalence Results.

	FC1-100		FC2-100		CNN1		CNN2	
	Result	Time	Result	Time	Result	Time	Result	Time
$\delta = 1.7$	SAT	90	SAT	1540	SAT	222	SAT	409
$\delta = 1.9$	SAT	117	Unkn	1800	SAT	219	SAT	426
$\delta = 2.1$	SAT	131	Unkn	1800	SAT	578	SAT	360
$\delta = 2.3$	SAT	159	Unkn	1800	SAT	254	Unkn	1800
$\delta = 5.0$	SAT	146	Unkn	1800	SAT	199	SAT	493
$\delta = 15$	SAT	117	Unkn	1800	SAT	162	SAT	796
$\delta = 20$	SAT	165	Unkn	1800	SAT	127	SAT	508

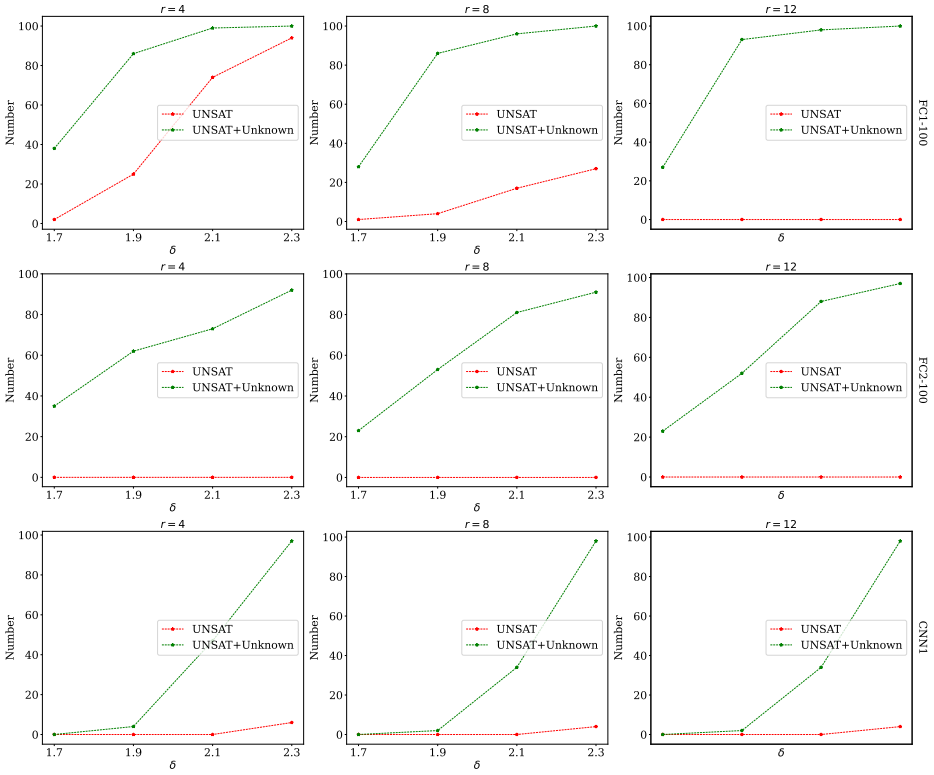

Fig. 3. Verification results for EQEV.

11 Conclusion

In this work, we propose a parallel method for verifying the equivalence between QNNs and their corresponding original real-valued NNs. To address the scalability issues arising from the involvement of two networks, our parallel approach employs a two-level hybrid parallel strategy and optimizes the process management method. Additionally, we propose two partition methods based on properties and outputs. Experimental results show that our equivalence verification tool is more efficient and can tackle more difficiult instances than previously existing approaches. Based on the analysis results from our tools, we found that the quantization schemes adopted in mainstream deep learning frameworks effectively maintain the equivalence within the data distribution space, but this is achieved at the cost of sacrificing equivalence outside of the data distribution. In future work, it would be interesting to formally analyze the equivalence between the original networks and the quantized neural networks in the context of applications, i.e. to investigate whether the results produced by the two networks are equivalent for specific application scenarios.

12 Acknowledgments

This work was funded in part by a Ford Alliance Project (199909), NSF (grant number 2211505), and the Stanford Center for AI Safety.

References

1. Fsd chip-tesla. `https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip` (2022)
2. Bunda, S., Spreeuwers, L.J., Zeinstra, C.G.: Sub-byte quantization of mobile face recognition convolutional neural networks. In: Brömme, A., Damer, N., Gomez-Barrero, M., Raja, K.B., Rathgeb, C., Sequeira, A.F., Todisco, M., Uhl, A. (eds.) Proceedings of the 21st International Conference of the Biometrics Special Interest Group, BIOSIG 2022, Darmstadt, Germany, September 14-16, 2022. LNI, vol. P-329, pp. 229–236. IEEE / Gesellschaft für Informatik e.V. (2022)
3. Cheng, C., Nührenberg, G., Ruess, H.: 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. Lecture Notes in Computer Science, vol. 10482, pp. 251–268. Springer (2017)
4. Ciresan, D.C., Giusti, A., Gambardella, L.M., Schmidhuber, J.: 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. pp. 2852–2860 (2012)
5. Devlin, J., Chang, M., Lee, K., Toutanova, K.: 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. pp. 4171–4186. Association for Computational Linguistics (2019)
6. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: 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 (2021)
7. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: 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. Lecture Notes in Computer Science, vol. 10811, pp. 121–138. Springer (2018)
8. Ehlers, R.: 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. Lecture Notes in Computer Science, vol. 10482, pp. 269–286. Springer (2017)
9. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints An Int. J. **23**(3), 296–309 (2018)
10. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: 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. pp. 3–18. IEEE Computer Society (2018)

11. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive computation and machine learning, MIT Press (2016)
12. Gurobi: A most powerful mathematical optimization solver (2018)
13. Han, S., Mao, H., Dally, W.J.: 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 (2016)
14. Henzinger, T.A., Lechner, M., Žikelić, D.: Scalable verification of quantized neural networks. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021. pp. 3787–3795. AAAI Press (2021)
15. Heule, M., Kullmann, O., Wieringa, S., Biere, A.: Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In: Eder, K., Lourenço, J., Shehory, O. (eds.) Hardware and Software: Verification and Testing - 7th International Haifa Verification Conference, HVC 2011, Haifa, Israel, December 6-8, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7261, pp. 50–65. Springer (2011)
16. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In: Creignou, N., Berre, D.L. (eds.) Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9710, pp. 228–245. Springer (2016)
17. Huang, P., Liu, M., Ge, C., Ma, F., Zhang, J.: Investigating the existence of orthogonal golf designs via satisfiability testing. In: Davenport, J.H., Wang, D., Kauers, M., Bradford, R.J. (eds.) Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation, ISSAC 2019, Beijing, China, July 15-18, 2019. pp. 203–210. ACM (2019)
18. Huang, P., Ma, F., Ge, C., Zhang, J., Zhang, H.: Investigating the existence of large sets of idempotent quasigroups via satisfiability testing. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10900, pp. 354–369. Springer (2018)
19. Huang, P., Wu, H., Yang, Y., Daukantas, I., Wu, M., Zhang, Y., Barrett, C.: Towards efficient verification of quantized neural networks. arXiv preprint arXiv:2312.12679 (2023)
20. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10426, pp. 3–29. Springer (2017)
21. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A.G., Adam, H., Kalenichenko, D.: 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. pp. 2704–2713. Computer Vision Foundation / IEEE Computer Society (2018)
22. Jia, K., Rinard, M.C.: 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 (2020)
23. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: 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. Lecture Notes in Computer Science, vol. 10426, pp. 97–117. Springer (2017)
24. 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., Barrett, C.W.: 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. Lecture Notes in Computer Science, vol. 11561, pp. 443–452. Springer (2019)
25. Kulkarni, U., M, M.S., Gurlahosur, S.V., Bhogar, G.: Quantization friendly mobilenet (qf-mobilenet) architecture for vision based applications on embedded platforms. Neural Networks **136**, 28–39 (2021)
26. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791, `https://doi.org/10.1109/5.726791`
27. Mirman, M., Gehr, T., Vechev, M.T.: 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. Proceedings of Machine Learning Research, vol. 80, pp. 3575–3583. PMLR (2018)
28. Mistry, S., Saha, I., Biswas, S.: An MILP encoding for efficient verification of quantized deep neural networks. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **41**(11), 4445–4456 (2022)
29. Paulsen, B., Wang, J., Wang, C.: Reludiff: differential verification of deep neural networks. In: Rothermel, G., Bae, D. (eds.) ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020. pp. 714–726. ACM (2020)
30. Paulsen, B., Wang, J., Wang, J., Wang, C.: NEURODIFF: scalable differential verification of neural networks using fine-grained approximation. In: 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020. pp. 784–796. IEEE (2020)
31. Raghunathan, A., Steinhardt, J., Liang, P.: 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. pp. 10900–10910 (2018)
32. Simonyan, K., Zisserman, A.: 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 (2015)
33. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 41:1–41:30 (2019)
34. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: 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. pp. 6369–6379 (2018)
35. Weng, T., Zhang, H., Chen, H., Song, Z., Hsieh, C., Daniel, L., Boning, D.S., Dhillon, I.S.: 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. Proceedings of Machine Learning Research, vol. 80, pp. 5273–5282. PMLR (2018)

36. Wong, E., Kolter, J.Z.: 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. Proceedings of Machine Learning Research, vol. 80, pp. 5283–5292. PMLR (2018)
37. Wu, H., Ozdemir, A., Zeljic, A., Julian, K., Irfan, A., Gopinath, D., Fouladi, S., Katz, G., Pasareanu, C.S., Barrett, C.W.: Parallelization techniques for verifying neural networks. In: 2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020. pp. 128–137. IEEE (2020)
38. Xu, H., Gao, Y., Yu, F., Darrell, T.: 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. pp. 3530–3538. IEEE Computer Society (2017)
39. Yang, Y., Lei, W., Huang, P., Cao, J., Li, J., Chua, T.: A dual prompt learning framework for few-shot dialogue state tracking. In: Ding, Y., Tang, J., Sequeda, J.F., Aroyo, L., Castillo, C., Houben, G. (eds.) Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023. pp. 1468–1477. ACM (2023)
40. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: 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. pp. 4944–4953 (2018)
41. Zhang, Y., Song, F., Sun, J.: Qebverif: Quantization error bound verification of neural networks. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13965, pp. 413–437. Springer (2023)
42. Zhang, Y., Zhao, Z., Chen, G., Song, F., Zhang, M., Chen, T., Sun, J.: 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. pp. 82:1–82:13. ACM (2022)