7-2023

QEBVerif: Quantization error bound verification of neural networks

Yedi ZHANG

Fu SONG

Jun SUN
Singapore Management University, junsun@smu.edu.sg

QEBVerif: Quantization Error Bound Verification of Neural Networks

Yedi Zhang[1], Fu Song[1,2,3(✉)], and Jun Sun[4]

[1] ShanghaiTech University, Shanghai 201210, China
songfu@shanghaitech.edu.cn
[2] Institute of Software, Chinese Academy of Sciences and University of Chinese Academy of Sciences, Beijing 100190, China
[3] Automotive Software Innovation Center, Chongqing 400000, China
[4] Singapore Management University, Singapore 178902, Singapore

Abstract. To alleviate the practical constraints for deploying deep neural networks (DNNs) on edge devices, quantization is widely regarded as one promising technique. It reduces the resource requirements for computational power and storage space by quantizing the weights and/or activation tensors of a DNN into lower bit-width fixed-point numbers, resulting in quantized neural networks (QNNs). While it has been empirically shown to introduce minor accuracy loss, critical verified properties of a DNN might become invalid once quantized. Existing verification methods focus on either individual neural networks (DNNs or QNNs) or quantization error bound for *partial* quantization. In this work, we propose a quantization error bound verification method, named QEBVerif, where both weights and activation tensors are quantized. QEBVerif consists of two parts, i.e., a differential reachability analysis (DRA) and a mixed-integer linear programming (MILP) based verification method. DRA performs difference analysis between the DNN and its quantized counterpart layer-by-layer to compute a tight quantization error interval efficiently. If DRA fails to prove the error bound, then we encode the verification problem into an equivalent MILP problem which can be solved by off-the-shelf solvers. Thus, QEBVerif is sound, complete, and reasonably efficient. We implement QEBVerif and conduct extensive experiments, showing its effectiveness and efficiency.

1 Introduction

In the past few years, the development of deep neural networks (DNNs) has grown at an impressive pace owing to their outstanding performance in solving various complicated tasks [23,28]. However, modern DNNs are often large in size and contain a great number of 32-bit floating-point parameters to achieve competitive performance. Thus, they often result in high computational costs and excessive storage requirements, hindering their deployment on resource-constrained embedded devices, e.g., edge devices. A promising solution is to quantize the weights and/or activation tensors as fixed-point numbers of lower

C. Enea and A. Lal (Eds.): CAV 2023, LNCS 13965, pp. 413–437, 2023.
https://doi.org/10.1007/978-3-031-37703-7_20

bit-width [17,21,25,35]. For example, TensorFlow Lite [18] supports quantization of weights and/or activation tensors to reduce the model size and latency, and Tesla FSD-chip [61] stores all the data and weights of a network in the form of 8-bit integers.

In spite of the empirically impressive results which show there is only minor accuracy loss, quantization does not necessarily preserve properties such as robustness [16]. Even worse, input perturbation can be amplified by quantization [11,36], worsening the robustness of quantized neural networks (QNNs) compared to their DNN counterparts. Indeed, existing neural network quantization methods focus on minimizing its impact on model accuracy (e.g., by formulating it as an optimization problem that aims to maximize the accuracy [27,43]). However, they cannot guarantee that the final quantization error is always lower than a given error bound, especially when some specific safety-critical input regions are concerned. This is concerning as such errors may lead to catastrophes when the quantized networks are deployed in safety-critical applications [14,26]. Furthermore, analyzing (in particular, quantifying) such errors can also help us understand how quantization affect the network behaviors [33], and provide insights on, for instance, how to choose appropriate quantization bit sizes without introducing too much error. Therefore, a method that soundly quantifies the errors between DNNs and their quantized counterparts is highly desirable.

There is a large and growing body of work on developing verification methods for DNNs [2,12,13,15,19,24,29,30,32,37,38,51,54,55,58–60,62] and QNNs [1,3,16,22,46,66,68], aiming to establish a formal guarantee on the network behaviors. However, all the above-mentioned methods focus exclusively on verifying individual neural networks. Recently, Paulsen et al. [48,49] proposed differential verification methods, aimed to establish formal guarantees on the difference between two DNNs. Specifically, given two DNNs $\mathcal{N}_1$ and $\mathcal{N}_2$ with the same network topology and inputs, they try to prove that $|\mathcal{N}_1(\mathbf{x}) - \mathcal{N}_2(\mathbf{x})| < \epsilon$ for all possible inputs $\mathbf{x} \in \mathcal{X}$, where $\mathcal{X}$ is the interested input region. They presented fast and sound difference propagation techniques followed by a refinement of the input region until the property can be successfully verified, i.e., the property is either proved or falsified by providing a counterexample. This idea has been extended to handle recurrent neural networks (RNNs) [41] though the refinement is not considered therein. Although their methods [41,48,49] can be used to analyze the error bound introduced by quantizing weights (called *partially* QNNs), they are not complete and cannot handle the cases where both the weights and activation tensors of a DNN are quantized to lower bit-width fixed-point numbers (called *fully* QNNs). We remark that fully QNN can significantly reduce the energy-consumption (floating-point operations consume much more energy than integer-only operations) [61].

Main Contributions. We propose a sound and complete **Q**uantization **E**rror **B**ound **Verif**ication method (QEBVerif) to efficiently and effectively verify if the quantization error of a *fully* QNN w.r.t. an input region and its original DNN is always lower than an error bound (a.k.a. robust error bound [33]). QEBVerif

first conducts a novel reachability analysis to quantify the quantization errors, which is referred to as *differential reachability analysis* (DRA). Such an analysis yields two results: (1) *Proved*, meaning that the quantization error is proved to be always less than the given error bound; or (2) *Unknown*, meaning that it fails to prove the error bound, possibly due to a conservative approximation of the quantization error. If the outcome is *Unknown*, we further encode this quantization error bound verification problem into an equivalent mixed-integer linear programming (MILP) problem, which can be solved by off-the-shelf solvers.

There are two main technical challenges that must be addressed for DRA. First, the activation tensors in a fully QNN are discrete values and contribute additional rounding errors to the final quantization errors, which are hard to propagate symbolically and make it difficult to establish relatively accurate difference intervals. Second, much more activation-patterns (i.e., $3 \times 6 = 18$) have to consider in a forward propagation, while 9 activation-patterns are sufficient in [48,49], where an activation-pattern indicates the status of the output range of a neuron. A neuron in a DNN under an input region has 3 patterns: always-active (i.e., output ≥ 0), always-inactive (i.e., output < 0), or both possible. A neuron in a QNN has 6 patterns due to the clamp function (cf. Definition 2). We remark that handling these different combinations efficiently and soundly is highly nontrivial. To tackle the above challenges, we propose sound transformations for the affine and activation functions to propagate quantization errors of two networks layer-by-layer. Moreover, for the affine transformation, we provide two alternative solutions: *interval-based* and *symbolic-based*. The former directly computes sound difference intervals via interval analysis [42], while the latter leverages abstract interpretation [10] to compute sound and symbolic difference intervals, using the polyhedra abstract domain. In comparison, the symbolic-based one is usually more accurate but less efficient than the interval-based one. Note that though existing tools can obtain quantization error intervals by independently computing the output intervals of two networks followed by interval subtractions, such an approach is often too conservative.

To resolve those problems that cannot be proved via our DRA, we resort to the sound and complete MILP-based verification method. Inspired by the MILP encoding of DNN and QNN verification [39,40,68], we propose a novel MILP encoding for verifying quantization error bounds. QEBVerif represents both the computations of the QNN and the DNN in mixed-integer linear constraints which are further simplified using their own output intervals. Moreover, we also encode the output difference intervals of hidden neurons from our DRA as mixed-integer linear constraints to boost the verification.

We implement our method as an end-to-end tool and use Gurobi [20] as our back-end MILP solver. We extensively evaluate it on a large set of verification tasks using neural networks for ACAS Xu [26] and MNIST [31], where the number of neurons varies from 310 to 4890, the number of bits for quantizing weights and activation tensors ranges from 4 to 10 bits, and the number of bits for quantizing inputs is fixed to 8 bits. For DRA, we compare QEBVerif with a naive method that first independently computes the output intervals of DNNs

and QNNs using the existing state-of-the-art (symbolic) interval analysis [22,55], and then conducts an interval subtraction. The experimental results show that both our interval- and symbolic-based approaches are much more accurate and can successfully verify much more tasks without the MILP-based verification. We also find that the quantization error interval returned by DRA is getting tighter with the increase of the quantization bit size. The experimental results also confirm the effectiveness of our MILP-based verification method, which can help verify many tasks that cannot be solved by DRA solely. Finally, our results also allow us to study the potential correlation of quantization errors and robustness for QNNs using QEBVerif.

We summarize our contributions as follows:

- We introduce the first sound, complete, and reasonably efficient quantization error bound verification method QEBVerif for fully QNNs by cleverly combining novel DRA and MILP-based verification methods;
- We propose a novel DRA to compute sound and tight quantization error intervals accompanied by an abstract domain tailored to QNNs, which can significantly and soundly tighten the quantization error intervals;
- We implement QEBVerif as an end-to-end open-source tool [64] and conduct an extensive evaluation on various verification tasks, demonstrating its effectiveness and efficiency.

The source code of our tool and benchmarks are available at https://github.com/S3L-official/QEBVerif. Missing proofs, more examples, and experimental results can be found in [65].

2 Preliminaries

We denote by $\mathbb{R}, \mathbb{Z}, \mathbb{N}$ and $\mathbb{B}$ the sets of real-valued numbers, integers, natural numbers, and Boolean values, respectively. Let $[n]$ denote the integer set $\{1, \ldots, n\}$ for given $n \in \mathbb{N}$. We use **BOLD UPPERCASE** (e.g., $\mathbf{W}$) and **bold lowercase** (e.g., $\mathbf{x}$) to denote matrices and vectors, respectively. We denote by $\mathbf{W}_{i,j}$ the j-entry in the i-th row of the matrix $\mathbf{W}$, and by $\mathbf{x}_i$ the i-th entry of the vector $\mathbf{x}$. Given a matrix $\mathbf{W}$ and a vector $\mathbf{x}$, we use $\widehat{\mathbf{W}}$ and $\hat{\mathbf{x}}$ (resp. $\widetilde{\mathbf{W}}$ and $\tilde{\mathbf{x}}$) to denote their quantized/integer (resp. fixed-point) counterparts.

2.1 Neural Networks

A deep neural network (DNN) consists of a sequence of layers, where the first layer is the *input layer*, the last layer is the *output layer* and the others are called *hidden layers*. Each layer contains one or more neurons. A DNN is *feed-forward* if all the neurons in each non-input layer only receive inputs from the neurons in the preceding layer.

Definition 1 (Feed-forward Deep Neural Network). *A feed-forward DNN $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^s$ with d layers can be seen as a composition of d functions such that $\mathcal{N} = l_d \circ l_{d-1} \circ \cdots \circ l_1$. Then, given an input $\mathbf{x} \in \mathbb{R}^n$, the output of the DNN $\mathbf{y} = \mathcal{N}(\mathbf{x})$ can be obtained by the following recursive computation:*

- *Input layer* $l_1 : \mathbb{R}^n \to \mathbb{R}^{n_1}$ *is the identity function, i.e.,* $\mathbf{x}^1 = l_1(\mathbf{x}) = \mathbf{x}$*;*
- *Hidden layer* $l_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ *for* $2 \leq i \leq d-1$ *is the function such that* $\mathbf{x}^i = l_i(\mathbf{x}^{i-1}) = \phi(\mathbf{W}^i\mathbf{x}^{i-1} + \mathbf{b}^i)$*;*
- *Output layer* $l_d : \mathbb{R}^{n_{d-1}} \to \mathbb{R}^s$ *is the function such that* $\mathbf{y} = \mathbf{x}^d = l_d(\mathbf{x}^{d-1}) = \mathbf{W}^d\mathbf{x}^{d-1} + \mathbf{b}^d$*.*

where $n_1 = n$*,* $\mathbf{W}^i$ *and* $\mathbf{b}^i$ *are the weight matrix and bias vector in the* i*-th layer, and* $\phi(\cdot)$ *is the activation function which acts element-wise on an input vector.*

In this work, we focus on feed-forward DNNs with the most commonly used activation functions: the rectified linear unit (ReLU) function, defined as $\text{ReLU}(x) = \max(x, 0)$. We also use n_d to denote the output dimension s.

A quantized neural network (QNN) is structurally similar to its real-valued counterpart, except that all the parameters, inputs of the QNN, and outputs of all the hidden layers are quantized into integers according to the given quantization scheme. Then, the computation over real-valued arithmetic in a DNN can be replaced by the computation using integer arithmetic, or equally, fixed-point arithmetic. In this work, we consider the most common quantization scheme, i.e., symmetric uniform quantization [44]. We first give the concept of quantization configuration which effectively defines a quantization scheme.

A *quantization configuration* $\mathcal{C}$ is a tuple $\langle \tau, Q, F \rangle$, where Q and F are the total bit size and the fractional bit size allocated to a value, respectively, and $\tau \in \{+, \pm\}$ indicates if the quantized value is unsigned or signed. Given a real number $x \in \mathbb{R}$ and a quantization configuration $\mathcal{C} = \langle \tau, Q, F \rangle$, its quantized integer counterpart $\hat{x}$ and the fixed-point counterpart $\tilde{x}$ under the symmetric uniform quantization scheme are:

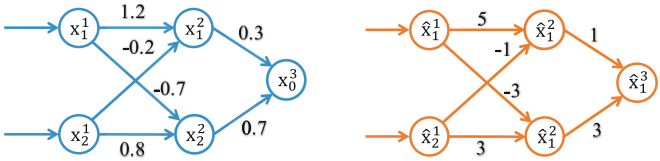
$$\hat{x} = \text{clamp}(\lfloor 2^F \cdot x \rceil,\ \mathcal{C}^{\text{lb}},\ \mathcal{C}^{\text{ub}}) \quad \text{and} \quad \tilde{x} = \hat{x}/2^F$$

where $\mathcal{C}^{\text{lb}} = 0$ and $\mathcal{C}^{\text{ub}} = 2^Q - 1$ if $\tau = +$, $\mathcal{C}^{\text{lb}} = -2^{Q-1}$ and $\mathcal{C}^{\text{ub}} = 2^{Q-1} - 1$ otherwise, and $\lfloor \cdot \rceil$ is the round-to-nearest integer operator. The *clamping function* $\text{clamp}(x, a, b)$ with a lower bound a and an upper bound b is defined as:

$$\text{clamp}(x, a, b) = \begin{cases} a, & \text{if } x < a; \\ x, & \text{if } a \leq x \leq b; \\ b, & \text{if } x > b. \end{cases}$$

Definition 2 (Quantized Neural Network). *Given quantization configurations for the weights, biases, output of the input layer and each hidden layer as* $\mathcal{C}_w = \langle \tau_w, Q_w, F_w \rangle$*,* $\mathcal{C}_b = \langle \tau_b, Q_b, F_b \rangle$*,* $\mathcal{C}_{in} = \langle \tau_{in}, Q_{in}, F_{in} \rangle$*,* $\mathcal{C}_h = \langle \tau_h, Q_h, F_h \rangle$*, the quantized version (i.e., QNN) of a DNN* $\mathcal{N}$ *with* d *layers is a function* $\widehat{\mathcal{N}} : \mathbb{Z}^n \to \mathbb{R}^s$ *such that* $\widehat{\mathcal{N}} = \hat{l}_d \circ \hat{l}_{d-1} \circ \cdots \circ \hat{l}_1$*. Then, given a quantized input* $\hat{\mathbf{x}} \in \mathbb{Z}^n$*, the output of the QNN* $\hat{\mathbf{y}} = \widehat{\mathcal{N}}(\hat{\mathbf{x}})$ *can be obtained by the following recursive computation:*

- *Input layer* $\hat{l}_1 : \mathbb{Z}^n \to \mathbb{Z}^{n_1}$ *is the identity function, i.e.,* $\hat{\mathbf{x}}^1 = \hat{l}_1(\hat{\mathbf{x}}) = \hat{\mathbf{x}}$*;*
- *Hidden layer* $\hat{l}_i : \mathbb{Z}^{n_{i-1}} \to \mathbb{Z}^{n_i}$ *for* $2 \leq i \leq d-1$ *is the function such that for each* $j \in [n_i]$*,*



(a) DNN $\mathcal{N}_e$. (b) QNN $\widehat{\mathcal{N}}_e$.

Fig. 1. A 3-layer DNN $\mathcal{N}_e$ and its quantized version $\widehat{\mathcal{N}}_e$.

$$\hat{\mathbf{x}}_j^i = \text{clamp}(\lfloor 2^{F_i}\widehat{\mathbf{W}}_{j,:}^i \cdot \hat{\mathbf{x}}^{i-1} + 2^{F_h-F_b}\hat{\mathbf{b}}_j^i \rceil, 0,\ \mathcal{C}_h^{\text{ub}}),$$

where F_i *is* $F_h - F_w - F_{in}$ *if* $i = 2$, *and* $-F_w$ *otherwise;*

– *Output layer* $\hat{l}_d : \mathbb{Z}^{n_{d-1}} \to \mathbb{R}^s$ *is the function such that* $\hat{\mathbf{y}} = \hat{\mathbf{x}}^d = \hat{l}_d(\hat{\mathbf{x}}^{d-1}) = 2^{-F_w}\widehat{\mathbf{W}}^d\hat{\mathbf{x}}^{d-1} + 2^{F_h-F_b}\hat{\mathbf{b}}^d$.

where for every $2 \le i \le d$ *and* $k \in [n_{i-1}]$, $\widehat{\mathbf{W}}_{j,k}^i = \text{clamp}(\lfloor 2^{F_w}\mathbf{W}_{j,k}^i \rceil, \mathcal{C}_w^{\text{lb}}, \mathcal{C}_w^{\text{ub}})$ *is the quantized weight and* $\hat{\mathbf{b}}_j^i = \text{clamp}(\lfloor 2^{F_b}\mathbf{b}_j^i \rceil, \mathcal{C}_b^{\text{lb}}, \mathcal{C}_b^{\text{ub}})$ *is the quantized bias.*

We remark that 2^{F_i} and $2^{F_h-F_b}$ in Definition 2 are used to align the precision between the inputs and outputs of hidden layers, and F_i for $i = 2$ and $i > 2$ because quantization bit sizes for the outputs of the input layer and hidden layers can be different.

2.2 Quantization Error Bound and Its Verification Problem

We now give the formal definition of the quantization error bound verification problem considered in this work as follows.

Definition 3 (Quantization Error Bound). *Given a DNN* $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^s$, *the corresponding QNN* $\widehat{\mathcal{N}} : \mathbb{Z}^n \to \mathbb{R}^s$, *a quantized input* $\hat{\mathbf{x}} \in \mathbb{Z}^n$, *a radius* $r \in \mathbb{N}$ *and an error bound* $\epsilon \in \mathbb{R}$. *The QNN* $\widehat{\mathcal{N}}$ *has a quantization error bound of* ϵ *w.r.t. the input region* $R(\hat{\mathbf{x}}, r) = \{\hat{\mathbf{x}}' \in \mathbb{Z}^n \mid ||\hat{\mathbf{x}}' - \hat{\mathbf{x}}||_\infty \le r\}$ *if for every* $\hat{\mathbf{x}}' \in R(\hat{\mathbf{x}}, r)$, *we have* $||2^{-F_h}\widehat{\mathcal{N}}(\hat{\mathbf{x}}') - \mathcal{N}(\mathbf{x}')||_\infty < \epsilon$, *where* $\mathbf{x}' = \hat{\mathbf{x}}'/(\mathcal{C}_{in}^{\text{ub}} - \mathcal{C}_{in}^{\text{lb}})$.

Intuitively, quantization-error-bound is the bound of the output difference of the DNN and its quantized counterpart for all the inputs in the input region. In this work, we obtain the input for DNN via dividing $\hat{\mathbf{x}}'$ by $(\mathcal{C}_{in}^{\text{ub}} - \mathcal{C}_{in}^{\text{lb}})$ to allow input normalization. Furthermore, 2^{-F_h} is used to align the precision between the outputs of QNN and DNN.

Example 1. Consider the DNN $\mathcal{N}_e$ with 3 layers (one input layer, one hidden layer, and one output layer) given in Fig. 1, where weights are associated with the edges and all the biases are 0. The quantization configurations for the weights, the output of the input layer and hidden layer are $\mathcal{C}_w = \langle \pm, 4, 2 \rangle$, $\mathcal{C}_{in} = \langle +, 4, 4 \rangle$ and $\mathcal{C}_h = \langle +, 4, 2 \rangle$. Its QNN $\widehat{\mathcal{N}}_e$ is shown in Fig. 1.

Given a quantized input $\hat{\mathbf{x}} = (9, 6)$ and a radius $r = 1$, the input region for QNN $\widehat{\mathcal{N}}_e$ is $R((9,6),1) = \{(x,y) \in \mathbb{Z}^2 \mid 8 \leq x \leq 10, 5 \leq y \leq 7\}$. Since $\mathcal{C}_{in}^{ub} = 15$ and $\mathcal{C}_{in}^{lb} = 0$, by Definitions 1, 2, and 3, we have the maximum quantization error as $\max(2^{-2}\widehat{\mathcal{N}}_e(\hat{\mathbf{x}}') - \mathcal{N}_e(\hat{\mathbf{x}}'/15)) = 0.067$ for $\hat{\mathbf{x}}' \in R((9,6),1)$. Then, $\widehat{\mathcal{N}}_e$ has a quantization error bound of ϵ w.r.t. input region $R((9,6),1)$ for any $\epsilon > 0.067$.

We remark that if only weights are quantized and the activation tensors are floating-point numbers, the maximal quantization error of $\widehat{\mathcal{N}}_e$ for the input region $R((9,6),1)$ is 0.04422, which implies that existing methods [48,49] cannot be used to analyze the error bound for a fully QNN.

In this work, we focus on the quantization error bound verification problem for classification tasks. Specifically, for a classification task, we only focus on the output difference of the predicted class instead of all the classes. Hence, given a DNN $\mathcal{N}$, a corresponding QNN $\widehat{\mathcal{N}}$, a quantized input $\hat{\mathbf{x}}$ which is classified to class g by the DNN $\mathcal{N}$, a radius r and an error bound ϵ, the quantization error bound property $P(\mathcal{N}, \widehat{\mathcal{N}}, \hat{\mathbf{x}}, r, \epsilon)$ for a classification task can be defined as follows:

$$\bigwedge_{\hat{\mathbf{x}}' \in R(\hat{\mathbf{x}},r)} \left(|2^{-F_h}\widehat{\mathcal{N}}(\hat{\mathbf{x}}')_g - \mathcal{N}(\mathbf{x}')_g| < \epsilon\right) \wedge \left(\mathbf{x}' = \hat{\mathbf{x}}'/(\mathcal{C}_{in}^{\text{ub}} - \mathcal{C}_{in}^{\text{lb}})\right)$$

Note that $\mathcal{N}(\cdot)_g$ denotes the g-th entry of the vector $\mathcal{N}(\cdot)$.

2.3 DeepPoly

We briefly recap DeepPoly [55], which will be leveraged in this work for computing the output of each neuron in a DNN.

The core idea of DeepPoly is to give each neuron an abstract domain in the form of a linear combination of the variables preceding the neuron. To achieve this, each hidden neuron $\mathbf{x}_j^i$ (the j-th neuron in the i-th layer) in a DNN is seen as two nodes $\mathbf{x}_{j,0}^i$ and $\mathbf{x}_{j,1}^i$, such that $\mathbf{x}_{j,0}^i = \sum_{k=1}^{n_{i-1}} \mathbf{W}_{j,k}^i \mathbf{x}_{k,1}^{i-1} + \mathbf{b}_j^i$ (affine function) and $\mathbf{x}_{j,1}^i = \text{ReLU}(\mathbf{x}_{j,0}^i)$ (ReLU function). Then, the affine function is characterized as an abstract transformer using an upper polyhedral computation and a lower polyhedral computation in terms of the variables $\mathbf{x}_{k,1}^{i-1}$. Finally, it recursively substitutes the variables in the upper and lower polyhedral computations with the corresponding upper/lower polyhedral computations of the variables until they only contain the input variables from which the concrete intervals are computed.

Formally, the abstract element $\mathcal{A}_{j,s}^i$ for the node $\mathbf{x}_{j,s}^i$ ($s \in \{0,1\}$) is a tuple $\mathcal{A}_{j,s}^i = \langle \mathbf{a}_{j,s}^{i,\leq}, \mathbf{a}_{j,s}^{i,\geq}, l_{j,s}^i, u_{j,s}^i \rangle$, where $\mathbf{a}_{j,s}^{i,\leq}$ and $\mathbf{a}_{j,s}^{i,\geq}$ are respectively the lower and upper polyhedral computations in the form of a linear combination of the variables $\mathbf{x}_{k,1}^{i-1}$'s if $s = 0$ or $\mathbf{x}_{k,0}^i$'s if $s = 1$, $l_{j,s}^i \in \mathbb{R}$ and $u_{j,s}^i \in \mathbb{R}$ are the concrete lower and upper bound of the neuron. Then, the concretization of the abstract element $\mathcal{A}_{j,s}^i$ is $\Gamma(\mathcal{A}_{j,s}^i) = \{x \in \mathbb{R} \mid \mathbf{a}_{j,s}^{i,\leq} \leq x \wedge x \leq \mathbf{a}_{j,s}^{i,\geq}\}$.

Concretely, $\mathbf{a}_{j,0}^{i,\leq}$ and $\mathbf{a}_{j,0}^{i,\geq}$ are defined as $\mathbf{a}_{j,0}^{i,\leq} = \mathbf{a}_{j,0}^{i,\geq} = \sum_{k=1}^{n_{i-1}} \mathbf{W}_{j,k}^i \mathbf{x}_{k,1}^{i-1} + \mathbf{b}_j^i$. Furthermore, we can repeatedly substitute every variable in $\mathbf{a}_{j,0}^{i,\leq}$ (resp. $\mathbf{a}_{j,0}^{i,\geq}$) with

its lower (resp. upper) polyhedral computation according to the coefficients until no further substitution is possible. Then, we can get a sound lower (resp. upper) bound in the form of a linear combination of the input variables based on which $l^i_{j,0}$ (resp. $u^i_{j,0}$) can be computed immediately from the given input region.

For ReLU function $\mathbf{x}^i_{j,1} = \text{ReLU}(\mathbf{x}^i_{j,0})$, there are three cases to consider of the abstract element $\mathcal{A}^i_{j,1}$:

- If $u^i_{j,0} \leq 0$, then $\mathbf{a}^{i,\leq}_{j,1} = \mathbf{a}^{i,\geq}_{j,1} = 0$, $l^i_{j,1} = u^i_{j,1} = 0$;
- If $l^i_{j,0} \geq 0$, then $\mathbf{a}^{i,\leq}_{j,1} = \mathbf{a}^{i,\leq}_{j,0}$, $\mathbf{a}^{i,\geq}_{j,1} = \mathbf{a}^{i,\geq}_{j,0}$, $l^i_{j,1} = l^i_{j,0}$ and $u^i_{j,1} = u^i_{j,0}$;
- If $l^i_{j,0} < 0 \wedge u^i_{j,0} > 0$, then $\mathbf{a}^{i,\geq}_{j,1} = \frac{u^i_{j,0}(\mathbf{x}^i_{j,0} - l^i_{j,0})}{u^i_{j,0} - l^i_{j,0}}$, $\mathbf{a}^{i,\leq}_{j,1} = \lambda \mathbf{x}^i_{j,0}$ where $\lambda \in \{0, 1\}$ such that the area of resulting shape by $\mathbf{a}^{i,\leq}_{j,1}$ and $\mathbf{a}^{i,\geq}_{j,1}$ is minimal, $l^i_{j,1} = \lambda l^i_{j,0}$ and $u^i_{j,1} = u^i_{j,0}$.

Note that DeepPoly also introduces transformers for other functions, such as sigmoid, tanh, and maxpool functions. In this work, we only consider DNNs with only ReLU as non-linear operators.

3 Methodology of QEBVerif

In this section, we first give an overview of our quantization error bound verification method, QEBVerif, and then give the detailed design of each component.

3.1 Overview of QEBVerif

An overview of QEBVerif is shown in Fig. 2. Given a DNN $\mathcal{N}$, its QNN $\widehat{\mathcal{N}}$, a quantization error bound ϵ and an input region consisting of a quantized input $\hat{\mathbf{x}}$ and a radius r, to verify the quantization error bound property $P(\mathcal{N}, \widehat{\mathcal{N}}, \hat{\mathbf{x}}, r, \epsilon)$, QEBVerif first performs a differential reachability analysis (DRA) to compute a sound output difference interval for the two networks. Note that, the difference intervals of all the neurons are also recorded for later use. If the output difference interval of the two networks is contained in $[-\epsilon, \epsilon]$, then the property is proved and QEBVerif outputs "Proved". Otherwise, QEBVerif leverages our MILP-based quantization error bound verification method by encoding the problem into an equivalent mixed integer linear programming (MILP) problem which can be solved by off-the-shelf solvers. To reduce the size of mixed integer linear constraints and boost the verification, QEBVerif independently applies symbolic interval analysis on the two networks based on which some activation patterns could be omitted. We further encode the difference intervals of all the neurons from DRA as mixed integer linear constraints and add them to the MILP problem. Though it increases the number of mixed integer linear constraints, it is very helpful for solving hard verification tasks. Therefore, the whole verification process is sound, complete yet reasonably efficient. We remark that the MILP-based verification method is often more time-consuming and thus the first step allows us to quickly verify many tasks first.

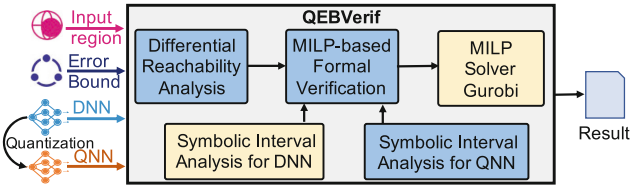


Fig. 2. An overview of QEBVerif.

3.2 Differential Reachability Analysis

Naively, one could use an existing verification tool in the literature to independently compute the output intervals for both the QNN and the DNN, and then compute their output difference directly by interval subtraction. However, such an approach would be ineffective due to the significant precision loss.

Recently, Paulsen et al. [48] proposed ReluDiff and showed that the accuracy of output difference for two DNNs can be greatly improved by propagating the difference intervals layer-by-layer. For each hidden layer, they first compute the output difference of affine functions (before applying the ReLU), and then they use a ReLU transformer to compute the output difference after applying the ReLU functions. The reason why ReluDiff outperforms the naive method is that ReluDiff first computes part of the difference before it accumulates. ReluDiff is later improved to tighten the approximated difference intervals [49]. However, as mentioned previously, they do not support *fully* quantified neural networks. Inspired by their work, we design a difference propagation algorithm for our setting. We use $S^{in}(\mathbf{x}_j^i)$ (resp. $S^{in}(\hat{\mathbf{x}}_j^i)$) to denote the interval of the j-th neuron in the i-th layer in the DNN (resp. QNN) before applying the ReLU function (resp. clamp function), and use $S(\mathbf{x}_j^i)$ (resp. $S(\hat{\mathbf{x}}_j^i)$) to denote the output interval after applying the ReLU function (resp. clamp function). We use δ_i^{in} (resp. δ_i) to denote the difference interval for the i-th layer before (resp. after) applying the activation functions, and use $\delta_{i,j}^{in}$ (resp. $\delta_{i,j}$) to denote the interval for the j-th neuron of the i-th layer. We denote by $\mathrm{LB}(\cdot)$ and $\mathrm{UB}(\cdot)$ the concrete lower and upper bounds accordingly.

Based on the above notations, we give our difference propagation in Algorithm 1. It works as follows. Given a DNN $\mathcal{N}$, a QNN $\widehat{\mathcal{N}}$ and a quantized input region $R(\hat{\mathbf{x}}, r)$, we first compute intervals $S^{in}(\mathbf{x}_j^i)$ and $S(\mathbf{x}_j^i)$ for neurons in $\mathcal{N}$ using *symbolic* interval analysis DeepPoly, and compute interval $S^{in}(\hat{\mathbf{x}}_j^i)$ and $S(\hat{\mathbf{x}}_j^i)$ for neurons in $\widehat{\mathcal{N}}$ using concrete interval analysis method [22]. Remark that no symbolic interval analysis for QNNs exists. By Definition 3, for each quantized input $\hat{\mathbf{x}}'$ for QNN, we obtain the input for DNN as $\mathbf{x}' = \hat{\mathbf{x}}'/(\mathcal{C}_{in}^{\mathrm{ub}} - \mathcal{C}_{in}^{\mathrm{lb}})$. After precision alignment, we get the input difference as $2^{-F_{in}}\hat{\mathbf{x}}' - \mathbf{x}' = (2^{-F_{in}} - 1/(\mathcal{C}_{in}^{\mathrm{ub}} - \mathcal{C}_{in}^{\mathrm{lb}}))\hat{\mathbf{x}}'$. Hence, given an input region, we get the output difference of the input layer: $\delta_1 = (2^{-F_{in}} - 1/(\mathcal{C}_{in}^{\mathrm{ub}} - \mathcal{C}_{in}^{\mathrm{lb}}))S(\hat{\mathbf{x}}^1)$. Then, we compute the output difference δ_i of each hidden layer iteratively by applying the affine transformer and activation transformer given in Algorithm 2 and Algorithm 3.

Algorithm 1: Forward Difference Propagation

Input : DNN $\mathcal{N}$, QNN $\widehat{\mathcal{N}}$, input region $R(\hat{x}, r)$
output: Output difference interval δ

Compute $S^{in}(\mathbf{x}_j^i)$ and $S(\mathbf{x}_j^i)$ for $i \in [d-1]$, $j \in [n_i]$ using DeepPoly;
Compute $S^{in}(\hat{\mathbf{x}}_j^i)$ and $S(\hat{\mathbf{x}}_j^i)$ for $i \in [d-1]$, $j \in [n_i]$ by applying interval analysis [22];
Initialize the difference: $\delta_1 = (2^{-F_{in}} - 1/(\mathcal{C}_{in}^{\text{ub}} - \mathcal{C}_{in}^{\text{lb}}))S(\hat{\mathbf{x}}^1)$;
for i in $2, \ldots, d-1$ **do** // propagate in hidden layers
  for j in $1, \ldots, n_i$ **do**
    $\Delta\mathbf{b}_j^i = 2^{-F_b}\hat{\mathbf{b}}_j^i - \mathbf{b}_j^i$; $\xi = 2^{-F_h - 1}$;
    $\delta_{i,j}^{in} = \text{AffTrs}(\mathbf{W}_{j,:}^i, 2^{-F_w}\widehat{\mathbf{W}}_{j,:}^i, \Delta\mathbf{b}_j^i, S(\mathbf{x}^{i-1}), \delta_{i-1}, \xi)$;
    $\delta_{i,j} = \text{ActTrs}(\delta_{i,j}^{in}, S^{in}(\mathbf{x}_j^i), 2^{-F_h}S^{in}(\hat{\mathbf{x}}_j^i))$;
// propagate in the output layer
for j in $1, \ldots, n_d$ **do**
  $\Delta\mathbf{b}_j^d = 2^{-F_b}\hat{\mathbf{b}}_j^d - \mathbf{b}_j^d$;
  $\delta_{d,j} = \delta_{d,j}^{in} = \text{AffTrs}(\mathbf{W}_{j,:}^d, 2^{-F_w}\widehat{\mathbf{W}}_{j,:}^d, \Delta\mathbf{b}_j^d, S(\mathbf{x}^{d-1}), \delta_{d-1}, 0)$;
return $(\delta_{i,j})_{2 \le i \le d, 1 \le j \le n_d}$;

Algorithm 2: AffTrs Function

Input : Weight vector $\mathbf{W}_{j,:}^i$, weight vector $\widetilde{\mathbf{W}}_{j,:}^i$, bias difference $\Delta\mathbf{b}_j^i$, neuron interval $S(\mathbf{x}^{i-1})$, difference interval δ_{i-1}, rounding error ξ
output: Difference interval $\delta_{i,j}^{in}$

$lb = \text{LB}(\widetilde{\mathbf{W}}_{j,:}^i \delta_{i-1} + (\widetilde{\mathbf{W}}_{j,:}^i - \mathbf{W}_{j,:}^i)S(\mathbf{x}^{i-1})) + \Delta\mathbf{b}_j^i - \xi$;
$ub = \text{UB}(\widetilde{\mathbf{W}}_{j,:}^i \delta_{i-1} + (\widetilde{\mathbf{W}}_{j,:}^i - \mathbf{W}_{j,:}^i)S(\mathbf{x}^{i-1})) + \Delta\mathbf{b}_j^i + \xi$;
return $[lb, ub]$;

Finally, we get the output difference for the output layer using only the affine transformer.

Affine Transformer. The difference before applying the activation function for the j-th neuron in the i-th layer is: $\delta_{i,j}^{in} = 2^{-F_h}\lfloor 2^{F_i}\widehat{\mathbf{W}}_{j,:}^i S(\hat{\mathbf{x}}^{i-1}) + 2^{F_h - F_b}\hat{\mathbf{b}}_j^i \rceil - \mathbf{W}_{j,:}^i S(\mathbf{x}^{i-1}) - \mathbf{b}_j^i$ where 2^{-F_h} is used to align the precision between the outputs of the two networks (cf. Sect. 2). Then, we soundly remove the rounding operators and give constraints for upper/lower bounds of $\delta_{i,j}^{in}$ as follows:

$$\text{UB}(\delta_{i,j}^{in}) \le \text{UB}(2^{-F_h}(2^{F_i}\widehat{\mathbf{W}}_{j,:}^i S(\hat{\mathbf{x}}^{i-1}) + 2^{F_h - F_b}\hat{\mathbf{b}}_j^i + 0.5) - \mathbf{W}_{j,:}^i S(\mathbf{x}^{i-1}) - \mathbf{b}^i)$$
$$\text{LB}(\delta_{i,j}^{in}) \ge \text{LB}(2^{-F_h}(2^{F_i}\widehat{\mathbf{W}}_{j,:}^i S(\hat{\mathbf{x}}^{i-1}) + 2^{F_h - F_b}\hat{\mathbf{b}}_j^i - 0.5) - \mathbf{W}_{j,:}^i S(\mathbf{x}^{i-1}) - \mathbf{b}^i)$$

Finally, we have $\text{UB}(\delta_{i,j}^{in}) \le \text{UB}\big(\widetilde{\mathbf{W}}_{j,:}^i S(\tilde{\mathbf{x}}^{i-1}) - \mathbf{W}_{j,:}^i S(\mathbf{x}^{i-1})\big) + \Delta\mathbf{b}_j^i + \xi$ and $\text{LB}(\delta_{i,j}^{in}) \ge \text{LB}\big(\widetilde{\mathbf{W}}_{j,:}^i S(\tilde{\mathbf{x}}^{i-1}) - \mathbf{W}_{j,:}^i S(\mathbf{x}^{i-1})\big) + \Delta\mathbf{b}_j^i - \xi$, which can be further reformulated as follows:

$$\text{UB}(\delta_{i,j}^{in}) \le \text{UB}\big(\widetilde{\mathbf{W}}_{j,:}^i \delta_{i-1} + \Delta\mathbf{W}_{j,:}^i S(\mathbf{x}^{i-1})\big) + \Delta\mathbf{b}_j^i + \xi$$
$$\text{LB}(\delta_{i,j}^{in}) \ge \text{LB}\big(\widetilde{\mathbf{W}}_{j,:}^i \delta_{i-1} + \Delta\mathbf{W}_{j,:}^i S(\mathbf{x}^{i-1})\big) + \Delta\mathbf{b}_j^i - \xi$$

where $S(\tilde{\mathbf{x}}^{i-1}) = 2^{-F_{in}}S(\hat{\mathbf{x}}^{i-1})$ if $i = 2$, and $2^{-F_h}S(\hat{\mathbf{x}}^{i-1})$ otherwise. $\widetilde{\mathbf{W}}_{j,:}^i = 2^{-F_w}\widehat{\mathbf{W}}_{j,:}^i$, $\Delta\mathbf{W}_{j,:}^i = \widetilde{\mathbf{W}}_{j,:}^i - \mathbf{W}_{j,:}^i$, $\Delta\mathbf{b}_j^i = 2^{-F_b}\hat{\mathbf{b}}_j^i - \mathbf{b}_j^i$ and $\xi = 2^{-F_h - 1}$.

Algorithm 3: ActTrs function

Input : Difference interval $\delta^{in}_{i,j}$, neuron interval $S^{in}(\mathbf{x}^i_j)$, neuron interval $S^{in}(\tilde{\mathbf{x}}^i_j)$, clamp upper bound t
output: Difference interval $\delta_{i,j}$

if $\mathsf{UB}(S^{in}(\mathbf{x}^i_j)) \leq 0$ **then** $lb = \text{clamp}(\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j)), 0, t)$; $ub = \text{clamp}(\mathsf{UB}(S^{in}(\tilde{\mathbf{x}}^i_j)), 0, t)$;
else if $\mathsf{LB}(S^{in}(\mathbf{x}^i_j)) \geq 0$ **then**
  if $\mathsf{UB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \leq t$ **and** $\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \geq 0$ **then** $lb = \mathsf{LB}(\delta^{in}_{i,j})$; $ub = \mathsf{UB}(\delta^{in}_{i,j})$;
  else if $\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \geq t$ **or** $\mathsf{UB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \leq 0$ **then**
    $lb = \text{clamp}(\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j)), 0, t) - \mathsf{UB}(S^{in}(\mathbf{x}^i_j))$;
    $ub = \text{clamp}(\mathsf{UB}(S^{in}(\tilde{\mathbf{x}}^i_j)), 0, t) - \mathsf{LB}(S^{in}(\mathbf{x}^i_j))$;
  else if $\mathsf{UB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \leq t$ **then**
    $lb = \max(-\mathsf{UB}(S^{in}(\mathbf{x}^i_j)), \mathsf{LB}(\delta^{in}_{i,j}))$; $ub = \max(-\mathsf{LB}(S^{in}(\mathbf{x}^i_j)), \mathsf{UB}(\delta^{in}_{i,j}))$;
  else if $\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \geq 0$ **then**
    $lb = \min(t - \mathsf{UB}(S^{in}(\mathbf{x}^i_j)), \mathsf{LB}(\delta^{in}_{i,j}))$; $ub = \min(t - \mathsf{LB}(S^{in}(\mathbf{x}^i_j)), \mathsf{UB}(\delta^{in}_{i,j}))$;
  else
    $lb = \max(-\mathsf{UB}(S^{in}(\mathbf{x}^i_j)), \min(t - \mathsf{UB}(S^{in}(\mathbf{x}^i_j)), \mathsf{LB}(\delta^{in}_{i,j})))$;
    $ub = \max(-\mathsf{LB}(S^{in}(\mathbf{x}^i_j)), \min(t - \mathsf{LB}(S^{in}(\mathbf{x}^i_j)), \mathsf{UB}(\delta^{in}_{i,j})))$;
else
  if $\mathsf{UB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \leq t$ **and** $\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \geq 0$ **then**
    $lb = \min(\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j)), \mathsf{LB}(\delta^{in}_{i,j}))$; $ub = \min(\mathsf{UB}(S^{in}(\tilde{\mathbf{x}}^i_j)), \mathsf{UB}(\delta^{in}_{i,j}))$;
  else if $\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \geq t$ **or** $\mathsf{UB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \leq 0$ **then**
    $lb = \text{clamp}(\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j)), 0, t) - \mathsf{UB}(S^{in}(\mathbf{x}^i_j))$; $ub = \text{clamp}(\mathsf{UB}(S^{in}(\tilde{\mathbf{x}}^i_j)), 0, t)$;
  else if $\mathsf{UB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \leq t$ **then**
    $lb = \max(\mathsf{LB}(\delta^{in}_{i,j}), -\mathsf{UB}(S^{in}(\mathbf{x}^i_j)))$; $ub = \min(\mathsf{UB}(\delta^{in}_{i,j}), \mathsf{UB}(S^{in}(\tilde{\mathbf{x}}^i_j)))$;
    if $\mathsf{UB}(\delta^{in}_{i,j}) \leq 0$ **then** $ub = 0$;
    if $\mathsf{LB}(\delta^{in}_{i,j}) \geq 0$ **then** $lb = 0$;
  else if $\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j)) \geq 0$ **then**
    $lb = \min(\mathsf{LB}(\delta^{in}_{i,j}), \mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j)), t - \mathsf{UB}(S^{in}(\mathbf{x}^i_j)))$; $ub = \min(\mathsf{UB}(\delta^{in}_{i,j}), t)$;
  else
    $lb = \min(t - \mathsf{UB}(S^{in}(\mathbf{x}^i_j)), 0, \max(\mathsf{LB}(\delta^{in}_{i,j}), -\mathsf{UB}(S^{in}(\mathbf{x}^i_j))))$;
    $ub = \text{clamp}(\mathsf{UB}(\delta^{in}_{i,j}), 0, t)$;
return $[lb, ub] \cap \big((S^{in}(\tilde{\mathbf{x}}^i_j) \cap [0, t]) - (S^{in}(\mathbf{x}^i_j) \cap [0, +\infty))\big)$;

Activation Transformer. Now we give our activation transformer in Algorithm 3 which computes the difference interval $\delta_{i,j}$ from the difference interval $\delta^{in}_{i,j}$. Note that, the neuron interval $S(\hat{\mathbf{x}}^i_j)$ for the QNN has already been converted to the fixed-point counterpart $S(\tilde{\mathbf{x}}^i_j) = 2^{-F_h} S(\hat{\mathbf{x}}^i_j)$ as an input parameter, as well as the clamping upper bound ($t = 2^{-F_h} \mathcal{C}^{\text{ub}}_h$). Different from ReluDiff [48] which focuses on the subtraction of two ReLU functions, here we investigate the subtraction of the clamping function and ReLU function.

Theorem 1. *If $\tau_h = +$, then Algorithm 1 is sound.*

Example 2. We exemplify Algorithm 1 using the networks $\mathcal{N}_e$ and $\widehat{\mathcal{N}}_e$ shown in Fig. 1. Given quantized input region $R((9, 6), 3)$ and the corresponding real-valued input region $R((0.6, 0.4), 0.2)$, we have $S(\hat{\mathbf{x}}^1_1) = [6, 12]$ and $S(\hat{\mathbf{x}}^1_2) = [3, 9]$.

First, we get $S^{in}(\mathbf{x}^2_1) = S(\mathbf{x}^2_1) = [0.36, 0.92]$, $S^{in}(\mathbf{x}^2_2) = [-0.4, 0.2]$, $S(\mathbf{x}^2_2) = [0, 0.2]$ based on DeepPoly and $S^{in}(\hat{\mathbf{x}}^2_1) = S(\hat{\mathbf{x}}^2_1) = [1, 4]$, $S^{in}(\hat{\mathbf{x}}^2_2) = [-2, 1]$, $S(\hat{\mathbf{x}}^2_2) = [0, 1]$ via interval analysis: $\text{LB}(S^{in}(\hat{\mathbf{x}}^2_1)) = \lfloor (5\text{LB}(\hat{\mathbf{x}}^1_1) - \text{UB}(\hat{\mathbf{x}}^1_2))/2^{-4} \rceil = 1$, $\text{UB}(S^{in}(\hat{\mathbf{x}}^2_1)) = \lfloor (5\text{UB}(\hat{\mathbf{x}}^1_1) - \text{LB}(\hat{\mathbf{x}}^1_2))/2^{-4} \rceil = 4$, $\text{LB}(S^{in}(\hat{\mathbf{x}}^2_2)) = \lfloor (-3\text{UB}(\hat{\mathbf{x}}^1_1) +$

$3\text{LB}(\hat{\mathbf{x}}_2^1))/2^{-4}\rceil = -2$, and $\text{UB}(S^{in}(\hat{\mathbf{x}}_2^2)) = \lfloor(-3\text{LB}(\hat{\mathbf{x}}_1^1) + 3\text{UB}(\hat{\mathbf{x}}_2^1))/2^{-4}\rceil = 1$. By Line 3 in Algorithm 1, we have $\delta_{1,1} = -\frac{1}{16\times 15}S(\hat{\mathbf{x}}_1^1) = [-0.05, -0.025]$, $\delta_{1,2} = -\frac{1}{16\times 15}S(\hat{\mathbf{x}}_2^1) = [-0.0375, -0.0125]$.

Then, we compute the difference interval before the activation functions. The rounding error is $\xi = 2^{-F_h-1} = 0.125$. We obtain the difference intervals $\delta_{2,1}^{in} = [-0.194375, 0.133125]$ and $\delta_{2,2}^{in} = [-0.204375, 0.123125]$ as follows based on Algorithm 2:

- $\text{LB}(\delta_{2,1}^{in}) = \text{LB}(\widetilde{\mathbf{W}}_{1,1}^1\delta_{1,1} + \widetilde{\mathbf{W}}_{1,2}^1\delta_{1,2} + \Delta\mathbf{W}_{1,1}^1 S(\mathbf{x}_1^1) + \Delta\mathbf{W}_{1,2}^1 S(\mathbf{x}_2^1)) - \xi = 1.25\times\text{LB}(\delta_{1,1}) - 0.25\times\text{UB}(\delta_{1,2}) + (1.25-1.2)\times\text{LB}(S(\mathbf{x}_1^1)) + (-0.25+0.2)\times\text{UB}(S(\mathbf{x}_2^1)) - 0.125$, $\text{UB}(\delta_{2,1}^{in}) = \text{UB}(\widetilde{\mathbf{W}}_{1,1}^1\delta_{1,1} + \widetilde{\mathbf{W}}_{1,2}^1\delta_{1,2} + \Delta\mathbf{W}_{1,1}^1 S(\mathbf{x}_1^1) + \Delta\mathbf{W}_{1,2}^1 S(\mathbf{x}_2^1)) + \xi = 1.25\times\text{UB}(\delta_{1,1}) - 0.25\times\text{LB}(\delta_{1,2}) + (1.25-1.2)\times\text{UB}(S(\mathbf{x}_1^1)) + (-0.25+0.2)\times\text{LB}(S(\mathbf{x}_2^1)) + 0.125$;
- $\text{LB}(\delta_{2,2}^{in}) = \text{LB}(\widetilde{\mathbf{W}}_{2,1}^1\delta_{1,1} + \widetilde{\mathbf{W}}_{2,2}^1\delta_{1,2} + \Delta\mathbf{W}_{2,1}^1 S(\mathbf{x}_1^1) + \Delta\mathbf{W}_{2,2}^1 S(\mathbf{x}_2^1)) - \xi = -0.75\times\text{UB}(\delta_{1,1}) + 0.75\times\text{LB}(\delta_{1,2}) + (-0.75+0.7)\times\text{UB}(S(\mathbf{x}_1^1)) + (0.75-0.8)\times\text{UB}(S(\mathbf{x}_2^1)) - 0.125$, $\text{UB}(\delta_{2,2}^{in}) = \text{UB}(\widetilde{\mathbf{W}}_{2,1}^1\delta_{1,1} + \widetilde{\mathbf{W}}_{2,2}^1\delta_{1,2} + \Delta\mathbf{W}_{2,1}^1 S(\mathbf{x}_1^1) + \Delta\mathbf{W}_{2,2}^1 S(\mathbf{x}_2^1)) + \xi = -0.75\times\text{LB}(\delta_{1,1}) + 0.75\times\text{UB}(\delta_{1,2}) + (-0.75+0.7)\times\text{LB}(S(\mathbf{x}_1^1)) + (0.75-0.8)\times\text{LB}(S(\mathbf{x}_2^1)) + 0.125$.

By Lines 20∼22 in Algorithm 3, we get the difference intervals after the activation functions for the hidden layer as: $\delta_{2,1} = \delta_{2,1}^{in} = [-0.194375, 0.133125]$, $\delta_{2,1} = [\max(\text{LB}(\delta_{2,2}^{in}), -\text{UB}(S^{in}(\mathbf{x}_2^2))), \min(\text{UB}(\delta_{2,2}^{in}), \text{UB}(S^{in}(\tilde{\mathbf{x}}_2^2)))] = [-0.2, 0.123125]$.

Next, we compute the output difference interval of the networks using Algorithm 2 again but with $\xi = 0$: $\text{LB}(\delta_{3,1}^{in}) = \text{LB}(\widetilde{\mathbf{W}}_{1,1}^2\delta_{2,1} + \widetilde{\mathbf{W}}_{1,2}^2\delta_{2,2} + \Delta\mathbf{W}_{1,1}^2 S(\mathbf{x}_1^2) + \Delta\mathbf{W}_{1,2}^2 S(\mathbf{x}_2^2)) = 0.25\times\text{LB}(\delta_{2,1}) + 0.75\times\text{LB}(\delta_{2,2}) + (0.25-0.3)\times\text{UB}(S(\mathbf{x}_1^2)) + (0.75-0.7)\times\text{LB}(S(\mathbf{x}_2^2))$, $\text{UB}(\delta_{3,2}^{in}) = \text{UB}(\widetilde{\mathbf{W}}_{1,1}^2\delta_{2,1} + \widetilde{\mathbf{W}}_{1,2}^2\delta_{2,2} + \Delta\mathbf{W}_{1,1}^2 S(\mathbf{x}_1^2) + \Delta\mathbf{W}_{1,2}^2 S(\mathbf{x}_2^2)) = 0.25\times\text{UB}(\delta_{2,1}) + 0.75\times\text{UB}(\delta_{2,2}) + (0.25-0.3)\times\text{LB}(S(\mathbf{x}_1^2)) + (0.75-0.7)\times\text{UB}(S(\mathbf{x}_2^2))$. Finally, the quantization error interval is [-0.24459375, 0.117625].

3.3 MILP Encoding of the Verification Problem

If DRA fails to prove the property, we encode the problem as an equivalent MILP problem. Specifically, we encode both the QNN and DNN as sets of (mixed integer) linear constraints, and quantize the input region as a set of integer linear constraints. We adopt the MILP encodings of DNNs [39] and QNNs [40] to transform the DNN and QNN into a set of linear constraints. We use (symbolic) intervals to further reduce the size of linear constraints similar to [39] while [40] did not. We suppose that the sets of constraints encoding the QNN, DNN, and quantized input region are $\Theta_{\widehat{\mathcal{N}}}$, $\Theta_{\mathcal{N}}$, and Θ_R, respectively. Next, we give the MILP encoding of the robust error bound property.

Recall that, given a DNN $\mathcal{N}$, an input region $R(\hat{\mathbf{x}}, r)$ such that $\mathbf{x}$ is classified to class g by $\mathcal{N}$, a QNN $\widehat{\mathcal{N}}$ has a quantization error bound ϵ w.r.t. $R(\hat{\mathbf{x}}, r)$ if for every $\hat{\mathbf{x}}' \in R(\hat{\mathbf{x}}, r)$, we have $|2^{-F_h}\widehat{\mathcal{N}}(\hat{\mathbf{x}}')_g - \mathcal{N}(\mathbf{x}')_g| < \epsilon$. Thus, it suffices to check if $|2^{-F_h}\widehat{\mathcal{N}}(\hat{\mathbf{x}}')_g - \mathcal{N}(\mathbf{x}')_g| \geq \epsilon$ for some $\hat{\mathbf{x}}' \in R(\hat{\mathbf{x}}, r)$.

Let $\hat{\mathbf{x}}_g^d$ (resp. $\mathbf{x}_g^d$) be the g-th output of $\widehat{\mathcal{N}}$ (resp. $\mathcal{N}$). We introduce a real-valued variable η and a Boolean variable v such that $\eta = \max(2^{-F_h}\hat{\mathbf{x}}_g^d - \mathbf{x}_g^d, 0)$ can be encoded by the set Θ_g of constraints with an extremely large number $\mathbf{M}$: $\Theta_g = \{\eta \geq 0,\ \eta \geq 2^{-F_h}\hat{\mathbf{x}}_g^d - \mathbf{x}_g^d,\ \eta \leq \mathbf{M} \cdot v,\ \eta \leq 2^{-F_h}\hat{\mathbf{x}}_g^d - \mathbf{x}_g^d + \mathbf{M} \cdot (1 - v)\}$. As a result, $|2^{-F_h}\hat{\mathbf{x}}_g^d - \mathbf{x}_g^d| \geq \epsilon$ iff the set of linear constraints $\Theta_\epsilon = \Theta_g \cup \{2\eta - (2^{-F_h}\hat{\mathbf{x}}_g^d - \mathbf{x}_g^d) \geq \epsilon\}$ holds.

Finally, the quantization error bound verification problem is equivalent to the solving of the constraints: $\Theta_P = \Theta_{\widehat{\mathcal{N}}} \cup \Theta_{\mathcal{N}} \cup \Theta_R \cup \Theta_\epsilon$. Remark that the output difference intervals of hidden neurons obtained from Algorithm 1 can be encoded as linear constraints which are added into the set Θ_P to boost the solving.

4 An Abstract Domain for Symbolic-Based DRA

While Algorithm 1 can compute difference intervals, the affine transformer explicitly adds a concrete rounding error interval to each neuron, which accumulates into a significant precision loss over the subsequent layers. To alleviate this problem, we introduce an abstract domain based on DeepPoly which helps to compute sound symbolic approximations for the lower and upper bounds of each difference interval, hence computing tighter difference intervals.

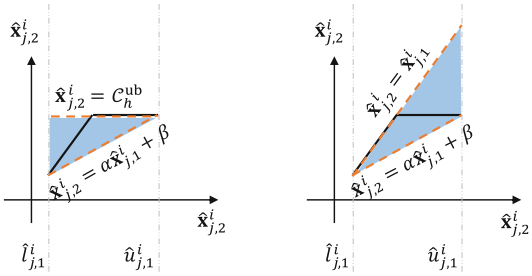
4.1 An Abstract Domain for QNNs

We first introduce transformers for affine transforms with rounding operators and clamp functions in QNNs. Recall that the activation function in a QNN $\widehat{\mathcal{N}}$ is also a min-ReLU function: $\min(\text{ReLU}(\lfloor\cdot\rceil), \mathcal{C}_h^{\text{ub}})$. Thus, we regard each hidden neuron $\hat{\mathbf{x}}_j^i$ in a QNN as three nodes $\hat{\mathbf{x}}_{j,0}^i$, $\hat{\mathbf{x}}_{j,1}^i$, and $\hat{\mathbf{x}}_{j,2}^i$ such that $\hat{\mathbf{x}}_{j,0}^i = \lfloor 2^{F_i}\sum_{k=1}^{n_{i-1}} \widehat{\mathbf{W}}_{j,k}^i \hat{\mathbf{x}}_{k,2}^{i-1} + 2^{F_h - F_b}\hat{\mathbf{b}}_j^i \rceil$ (affine function), $\hat{\mathbf{x}}_{j,1}^i = \max(\hat{\mathbf{x}}_{j,0}^i, 0)$ (ReLU function) and $\hat{\mathbf{x}}_{j,2}^i = \min(\hat{\mathbf{x}}_{j,1}^i, \mathcal{C}_h^{\text{ub}})$ (min function). We now give the abstract domain $\widehat{\mathcal{A}}_{j,p}^i = \langle \hat{\mathbf{a}}_{j,p}^{i,\leq}, \hat{\mathbf{a}}_{j,p}^{i,\geq}, \hat{l}_{j,p}^i, \hat{u}_{j,p}^i \rangle$ for each neuron $\hat{\mathbf{x}}_{j,p}^i$ ($p \in \{0, 1, 2\}$) in a QNN as follows.

Following DeepPoly, $\hat{\mathbf{a}}_{j,0}^{i,\leq}$ and $\hat{\mathbf{a}}_{j,0}^{i,\geq}$ for the affine function of $\hat{\mathbf{x}}_{j,0}^i$ with rounding operators are defined as $\hat{\mathbf{a}}_{j,0}^{i,\leq} = 2^{F_i}\sum_{k=1}^{n_{i-1}} \widehat{\mathbf{W}}_{j,k}^i \hat{\mathbf{x}}_{k,2}^{i-1} + 2^{F_h - F_b}\hat{\mathbf{b}}_j^i - 0.5$ and $\hat{\mathbf{a}}_{j,0}^{i,\geq} = 2^{F_i}\sum_{k=1}^{n_{i-1}} \widehat{\mathbf{W}}_{j,k}^i \hat{\mathbf{x}}_{k,2}^{i-1} + 2^{F_h - F_b}\hat{\mathbf{b}}_j^i + 0.5$. We remark that $+0.5$ and -0.5 here are added to soundly encode the rounding operators and have no effect on the perseverance of invariant since the rounding operators will add/subtract 0.5 at most to round each floating-point number into its nearest integer. The abstract transformer for the ReLU function $\mathbf{x}_{j,1}^i = \text{ReLU}(\mathbf{x}_{j,0}^i)$ is defined the same as DeepPoly.

For the min function $\hat{\mathbf{x}}_{j,2}^i = \min(\hat{\mathbf{x}}_{j,1}^i, \mathcal{C}_h^{\text{ub}})$, there are three cases for $\widehat{\mathcal{A}}_{j,2}^i$:

- If $\hat{l}_{j,1}^i \geq \mathcal{C}_h^{\text{ub}}$, then $\hat{\mathbf{a}}_{j,2}^{i,\leq} = \hat{\mathbf{a}}_{j,2}^{i,\geq} = \mathcal{C}_h^{\text{ub}}$, $\hat{l}_{j,2}^i = \hat{u}_{j,2}^i = \mathcal{C}_h^{\text{ub}}$;
- If $\hat{u}_{j,1}^i \leq \mathcal{C}_h^{\text{ub}}$, then $\hat{\mathbf{a}}_{j,2}^{i,\leq} = \hat{\mathbf{a}}_{j,1}^{i,\leq}$, $\hat{\mathbf{a}}_{j,2}^{i,\geq} = \hat{\mathbf{a}}_{j,1}^{i,\geq}$, $\hat{l}_{j,2}^i = \hat{l}_{j,1}^i$ and $\hat{u}_{j,2}^i = \hat{u}_{j,1}^i$;



(a) $(\gamma, \mu) = (0, \mathcal{C}^{\text{ub}}_h)$. (b) $(\gamma, \mu) = (1, 0)$.

Fig. 3. Convex approximation for the min function in QNNs, where Fig. 3(a) and Fig. 3(b) show the two ways where $\alpha = \frac{\mathcal{C}^{\text{ub}}_h - \hat{l}^i_{j,1}}{\hat{u}^i_{j,1} - \hat{l}^i_{j,1}}$ and $\beta = \frac{(\hat{u}^i_{j,1} - \mathcal{C}^{\text{ub}}_h)}{\hat{u}^i_{j,1} - \hat{l}^i_{j,1}}$.

– If $\hat{l}^i_{j,1} < \mathcal{C}^{\text{ub}}_h \wedge \hat{u}^i_{j,1} > \mathcal{C}^{\text{ub}}_h$, then $\hat{\mathbf{a}}^{i,\geq}_{j,2} = \lambda\hat{\mathbf{x}}^i_{j,1} + \mu$ and $\hat{\mathbf{a}}^{i,\leq}_{j,2} = \frac{\mathcal{C}^{\text{ub}}_h - \hat{l}^i_{j,1}}{\hat{u}^i_{j,1} - \hat{l}^i_{j,1}}\hat{\mathbf{x}}^i_{j,1} + \frac{(\hat{u}^i_{j,1} - \mathcal{C}^{\text{ub}}_h)}{\hat{u}^i_{j,1} - \hat{l}^i_{j,1}}\hat{l}^i_{j,1}$, where $(\lambda, \mu) \in \{(0, \mathcal{C}^{\text{ub}}_h), (1, 0)\}$ such that the area of resulting shape by $\hat{\mathbf{a}}^{i,\leq}_{j,2}$ and $\hat{\mathbf{a}}^{i,\geq}_{j,2}$ is minimal, $\hat{l}^i_{j,2} = \hat{l}^i_{j,1}$ and $\hat{u}^i_{j,2} = \lambda\hat{u}^i_{j,1} + \mu$. We show the two ways of approximation in Fig. 3.

Theorem 2. *The min abstract transformer preserves the following invariant:* $\Gamma(\widehat{\mathcal{A}}^i_{j,2}) \subseteq [\hat{l}^i_{j,2}, \hat{u}^i_{j,2}]$.

From our abstract domain for QNNs, we get a symbolic interval analysis, similar to the one for DNNs using DeepPoly, to replace Line 2 in Algorithm 1.

4.2 Symbolic Quantization Error Computation

Recall that to compute tight bounds of QNNs or DNNs via symbolic interval analysis, variables in upper and lower polyhedral computations are recursively substituted with the corresponding upper/lower polyhedral computations of variables until they only contain the input variables from which the concrete intervals are computed. This idea motivates us to design a symbolic difference computation approach for differential reachability analysis based on the abstract domain DeepPoly for DNNs and our abstract domain for QNNs.

Consider two hidden neurons $\mathbf{x}^i_{j,s}$ and $\hat{\mathbf{x}}^i_{j,s}$ from the DNN $\mathcal{N}$ and the QNN $\widehat{\mathcal{N}}$. Let $\mathcal{A}^{i,*}_{j,s} = \langle \mathbf{a}^{i,\leq,*}_{j,s}, \mathbf{a}^{i,\geq,*}_{j,s}, l^{i,*}_{j,s}, u^{i,*}_{j,s} \rangle$ and $\widehat{\mathcal{A}}^i_{j,p} = \langle \hat{\mathbf{a}}^{i,\leq,*}_{j,p}, \hat{\mathbf{a}}^{i,\geq,*}_{j,p}, \hat{l}^{i,*}_{j,p}, \hat{u}^{i,*}_{j,p} \rangle$ be their abstract elements, respectively, where all the polyhedral computations are linear combinations of the input variables of the DNN and QNN, respectively, i.e.,

– $\mathbf{a}^{i,\leq,*}_{j,s} = \sum_{k=1}^{m} \mathbf{w}^{l,*}_k \mathbf{x}^1_k + \mathbf{b}^{l,*}_j$, $\mathbf{a}^{i,\geq,*}_{j,s} = \sum_{k=1}^{m} \mathbf{w}^{u,*}_k \mathbf{x}^1_k + \mathbf{b}^{u,*}_j$;
– $\hat{\mathbf{a}}^{i,\leq,*}_{j,p} = \sum_{k=1}^{m} \hat{\mathbf{w}}^{l,*}_k \hat{\mathbf{x}}^1_k + \hat{\mathbf{b}}^{l,*}_j$, $\hat{\mathbf{a}}^{i,\geq,*}_{j,p} = \sum_{k=1}^{m} \hat{\mathbf{w}}^{u,*}_k \hat{\mathbf{x}}^1_k + \hat{\mathbf{b}}^{u,*}_j$.

Then, the sound lower bound $\Delta l^{i,*}_{j,s}$ and upper $\Delta u^{i,*}_{j,s}$ bound of the difference can be derived as follows, where $p = 2s$:

Table 1. Benchmarks for QNNs and DNNs on MNIST.

Arch	#Paras	QNNs				DNNs
		$Q=4$	$Q=6$	$Q=8$	$Q=10$	
P1: 1blk_100	≈ 79.5k	96.38%	96.79%	96.77%	96.74%	96.92%
P2: 2blk_100	≈ 89.6k	96.01%	97.04%	97.00%	97.02%	97.07%
P3: 3blk_100	≈ 99.7k	95.53%	96.66%	96.59%	96.68%	96.71%
P4: 2blk_512	≈ 669.7k	96.69%	97.41%	97.35%	97.36%	97.36%
P5: 4blk_1024	≈ 3,963k	97.71%	98.05%	98.01%	98.04%	97.97%

– $\Delta l^{i,*}_{j,s} = \mathrm{LB}(2^{-F_h}\hat{\mathbf{x}}^i_{j,p} - \mathbf{x}^i_{j,s}) = 2^{-F_h}\hat{\mathbf{a}}^{i,\leq,*}_{j,p} - \mathbf{a}^{i,\geq,*}_{j,s}$;
– $\Delta u^{i,*}_{j,s} = \mathrm{UB}(2^{-F_h}\hat{\mathbf{x}}^i_{j,p} - \mathbf{x}^i_{j,s}) = 2^{-F_h}\hat{\mathbf{a}}^{i,\geq,*}_{j,p} - \mathbf{a}^{i,\leq,*}_{j,s}$.

Given a quantized input $\hat{\mathbf{x}}$ of the QNN $\widehat{\mathcal{N}}$, the input difference of two networks is $2^{-F_{in}}\hat{\mathbf{x}} - \mathbf{x} = (2^{-F_{in}}\mathcal{C}^{\mathrm{ub}}_h - 1)\mathbf{x}$. Therefore, we have $\Delta^1_k = \tilde{\mathbf{x}}^1_k - \mathbf{x}^1_k = 2^{-F_{in}}\hat{\mathbf{x}}^1_k - \mathbf{x}^1_k = (2^{-F_{in}}\mathcal{C}^{\mathrm{ub}}_h - 1)\mathbf{x}$. Then, the lower bound of difference can be reformulated as follows which only contains the input variables of DNN $\mathcal{N}$: $\Delta l^{i,*}_{j,s} = \Delta \mathbf{b}^{l,*}_j + \sum_{k=1}^{m}(-\mathbf{w}^{u,*}_k + 2^{-F_{in}}\mathcal{C}^{\mathrm{ub}}_h\tilde{\mathbf{w}}^{l,*}_k)\mathbf{x}^1_k$, where $\Delta \mathbf{b}^{l,*}_j = 2^{-F_h}\hat{\mathbf{b}}^{l,*}_j - \mathbf{b}^{u,*}_j$, $F^* = F_{in} - F_h$, $\Delta^1_k = \tilde{\mathbf{x}}^1_k - \mathbf{x}^1_k$ and $\tilde{\mathbf{w}}^{l,*}_k = 2^{F^*}\hat{\mathbf{w}}^{l,*}_k$.

Similarly, we can reformulated the upper bound $\Delta u^{i,*}_{j,s}$ as follows using the input variables of the DNN: $\Delta u^{i,*}_{j,s} = \Delta \mathbf{b}^{u,*}_j + \sum_{k=1}^{m}(-\mathbf{w}^{l,*}_k + 2^{-F_{in}}\mathcal{C}^{\mathrm{ub}}_h\tilde{\mathbf{w}}^{u,*}_k)\mathbf{x}^1_k$, where $\Delta \mathbf{b}^{u,*}_j = 2^{-F_h}\hat{\mathbf{b}}^{u,*}_j - \mathbf{b}^{l,*}_j$, $F^* = F_{in} - F_h$, and $\tilde{\mathbf{w}}^{u,*}_k = 2^{F^*}\hat{\mathbf{w}}^{u,*}_k$.

Finally, we compute the concrete input difference interval $\delta^{in}_{i,j}$ based on the given input region as $\delta^{in}_{i,j} = [\mathrm{LB}(\Delta l^{i,*}_{j,0}), \mathrm{UB}(\Delta u^{i,*}_{j,0})]$, with which we can replace the AffTrs functions in Algorithm 1 directly. An illustrating example is given in [65].

5 Evaluation

We have implemented our method QEBVerif as an end-to-end tool written in Python, where we use Gurobi [20] as our back-end MILP solver. All floating-point numbers used in our tool are 32-bit. Experiments are conducted on a 96-core machine with Intel(R) Xeon(R) Gold 6342 2.80 GHz CPU and 1 TB main memory. We allow Gurobi to use up to 24 threads. The time limit for each verification task is 1 h.

Benchmarks. We first build 45 * 4 QNNs from the 45 DNNs of ACAS Xu [26], following a *post-training quantization scheme* [44] and using quantization configurations $\mathcal{C}_{in} = \langle \pm, 8, 8\rangle$, $\mathcal{C}_w = \mathcal{C}_b = \langle \pm, Q, Q-2\rangle$, $\mathcal{C}_h = \langle +, Q, Q-2\rangle$, where $Q \in \{4, 6, 8, 10\}$. We then train 5 DNNs with different architectures using the MNIST dataset [31] and build 5 * 4 QNNs following the same quantization scheme and quantization configurations except that we set $\mathcal{C}_{in} = \langle +, 8, 8\rangle$ and $\mathcal{C}_w = \langle \pm, Q, Q-1\rangle$ for each DNN trained on MNIST. Details on the networks

trained on the MNIST dataset are presented in Table 1. Column 1 gives the name and architecture of each DNN, where Ablk_B means that the network has A hidden layers with each hidden layer size B neurons, Column 2 gives the number of parameters in each DNN, and Columns 3–7 list the accuracy of these networks. Hereafter, we denote by Px-y (resp. Ax-y) the QNN using the architecture Px (using the x-th DNN) and quantization bit size $Q = y$ for MNIST (resp. ACAS Xu), and by Px-Full (resp. Ax-Full) the DNN of architecture Px for MNIST (resp. the x-th DNN in ACAS Xu).

5.1 Effectiveness and Efficiency of DRA

We first implement a naive method using existing state-of-the-art reachability analysis methods for QNNs and DNNs. Specifically, we use the symbolic interval analysis of DeepPoly [55] to compute the output intervals for a DNN, and use interval analysis of [22] to compute the output intervals for a QNN. Then, we compute quantization error intervals via interval subtraction. Note that no existing methods can directly verify quantization error bounds and the methods in [48,49] are not applicable. Finally, we compare the quantization error intervals computed by the naive method against DRA in QEBVerif, using DNNs Ax-Full, Py-Full and QNNs Ax-z, Py-z for $x = 1$, $y \in \{1, 2, 3, 4, 5\}$ and $z \in \{4, 6, 8, 10\}$. We use the same adversarial input regions (5 input points with radius $r = \{3, 6, 13, 19, 26\}$ for each point) as in [29] for ACAS Xu, and set the quantization error bound $\epsilon \in \{0.05, 0.1, 0.2, 0.3, 0.4\}$, i.e., resulting 25 tasks for each radius. For MNIST, we randomly select 30 input samples from the test set of MNIST and set radius $r = 3$ for each input sample and quantization error bound $\epsilon \in \{1, 2, 4, 6, 8\}$, resulting in a total of 150 tasks for each pair of DNN and QNN of same architecture for MNIST.

Table 2 reports the analysis results for ACAS Xu (above) and MNIST (below). Column 2 lists different analysis methods, where QEBVerif (Int) is Algorithm 1 and QEBVerif (Sym) uses a symbolic-based method for the affine transformation in Algorithm 1 (cf. Sect. 4.2). Columns (H_Diff) (resp. O_Diff) averagely give the sum ranges of the difference intervals of all the hidden neurons (resp. output neurons of the predicted class) for the 25 verification tasks for ACAS Xu and 150 verification tasks for MNIST. Columns (#S/T) list the number of tasks (#S) successfully proved by DRA and average computation time (T) in seconds, respectively, where the best ones (i.e., solving the most tasks) are highlighted in blue. Note that Table 2 only reports the number of true propositions proved by DRA while the exact number is unknown.

Unsurprisingly, QEBVerif (Sym) is less efficient than the others but is still in the same order of magnitude. However, we can observe that QEBVerif (Sym) solves the most tasks for both ACAS Xu and MNIST and produces the most accurate difference intervals of both hidden neurons and output neurons for almost all the tasks in MNIST, except for P1-8 and P1-10 where QEBVerif (Int) performs better on the intervals for the output neurons. We also find that QEBVerif (Sym) may perform worse than the naive method when the quantization bit size is small for ACAS Xu. It is because: (1) the rounding error added into

Table 2. Differential Reachability Analysis on ACAS Xu and MNIST.

Q	Method	$r = 3$			$r = 6$			$r = 13$			$r = 19$			$r = 26$		
		H_Diff	O_Diff	#S/T	H_Diff	O_Diff	#S/T	H_Diff	O_Diff	#S/T	H_Diff	O_Diff	#S/T	H_Diff	O_Diff	#S/T
4	Naive	270.5	0.70	15/0.47	423.7	0.99	9/0.52	1,182	4.49	0/0.67	6,110	50.91	0/0.79	18,255	186.6	0/0.81
	QEBVerif (Int)	270.5	0.70	15/0.49	423.4	0.99	9/0.53	1,181	4.46	0/0.70	6,044	50.91	0/0.81	17,696	186.6	0/0.85
	QEBVerif (Sym)	749.4	145.7	0/2.02	780.9	150.2	0/2.11	1,347	210.4	0/2.24	6,176	254.7	0/2.35	18,283	343.7	0/2.39
6	Naive	268.3	1.43	5/0.47	557.2	4.00	0/0.51	1,258	6.91	0/0.67	6,145	53.29	0/0.77	18,299	189.0	0/0.82
	QEBVerif (Int)	268.0	1.41	5/0.50	555.0	3.98	0/0.54	1,245	6.90	0/0.69	6,125	53.28	0/0.80	18,218	189.0	0/0.83
	QEBVerif (Sym)	299.7	2.58	10/1.48	365.1	3.53	9/1.59	1,032	7.65	5/1.91	5,946	85.46	4/2.15	18,144	260.5	0/2.27
8	Naive	397.2	3.57	0/0.47	587.7	5.00	0/0.51	1,266	7.90	0/0.67	6,160	54.27	0/0.78	18,308	190.0	0/0.81
	QEBVerif (Int)	388.4	3.56	0/0.49	560.1	5.00	0/0.53	1,222	7.89	0/0.69	6,103	54.27	0/0.79	18,212	190.0	0/0.83
	QEBVerif (Sym)	35.75	0.01	24/1.10	93.78	0.16	18/1.19	845.2	5.84	8/1.65	5,832	58.73	5/1.97	18,033	209.6	5/2.12
10	Naive	394.5	3.67	0/0.49	591.4	5.17	0/0.51	1,268	8.04	0/0.68	6,164	54.42	0/0.78	18,312	190.1	0/0.80
	QEBVerif (Int)	361.9	3.67	0/0.50	546.2	5.17	0/0.54	1,209	8.04	0/0.68	6,083	54.42	0/0.79	18,182	190.1	0/0.83
	QEBVerif (Sym)	15.55	0.01	25/1.04	54.29	0.06	22/1.15	764.6	4.53	9/1.52	5,780	57.21	5/1.91	18,011	228.7	5/2.08

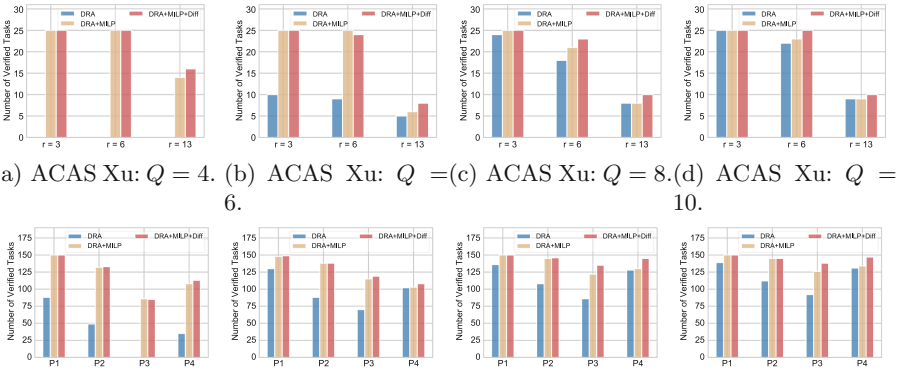
Q	Method	P1			P2			P3			P4			P5		
		H_Diff	O_Diff	#S/T	H_Diff	O_Diff	#S/T	H_Diff	O_Diff	#S/T	H_Diff	O_Diff	#S/T	H_Diff	O_Diff	#S/T
4	Naive	64.45	7.02	61/0.77	220.9	20.27	0/1.53	551.6	47.75	0/2.38	470.1	22.69	2/11.16	5,336	140.4	0/123.0
	QEBVerif (Int)	32.86	6.65	63/0.78	194.8	20.27	0/1.54	530.9	47.75	0/2.40	443.3	22.69	2/11.23	5,275	140.4	0/123.4
	QEBVerif (Sym)	32.69	3.14	88/1.31	134.9	7.11	49/2.91	313.8	14.90	1/5.08	365.2	11.11	35/22.28	1,864	50.30	1/310.2
6	Naive	68.94	7.89	66/0.77	249.5	24.25	0/1.52	616.2	54.66	0/2.38	612.2	31.67	1/11.18	7,399	221.0	0/125.4
	QEBVerif (Int)	10.33	2.19	115/0.78	89.66	12.81	14/1.54	466.0	52.84	0/2.39	307.6	20.22	5/11.28	7,092	221.0	0/125.1
	QEBVerif (Sym)	10.18	1.46	130/1.34	55.73	3.11	88/2.85	131.3	5.33	70/4.72	158.5	3.99	102/21.85	861.9	12.67	22/279.9
8	Naive	69.15	7.95	64/0.77	251.6	24.58	0/1.52	623.1	55.42	0/2.38	620.6	32.43	1/11.29	7,542	226.1	0/125.3
	QEBVerif (Int)	4.27	0.89	135/0.78	38.87	5.99	66/1.54	320.1	40.84	0/2.39	134.0	8.99	50/11.24	7,109	226.1	0/125.7
	QEBVerif (Sym)	4.13	1.02	136/1.35	34.01	2.14	108/2.82	82.90	3.48	86/4.61	96.26	2.39	128/21.45	675.7	6.20	27/273.6
10	Naive	69.18	7.96	65/0.77	252.0	24.63	0/1.52	624.0	55.55	0/2.36	620.4	32.40	1/11.19	7,559	226.9	0/124.2
	QEBVerif (Int)	2.72	0.56	139/0.78	25.39	4.15	79/1.53	260.9	34.35	0/2.40	84.12	5.75	73/11.26	7,090	226.9	0/125.9
	QEBVerif (Sym)	2.61	0.92	139/1.35	28.59	1.91	112/2.82	71.33	3.06	92/4.56	81.08	2.01	131/21.48	646.5	5.68	31/271.5

the abstract domain of the affine function in each hidden layer of QNNs is large due to the small bit size, and (2) such errors can accumulate and magnify layer by layer, in contrast to the naive approach where we directly apply the interval subtraction. We remark that symbolic-based reachability analysis methods for DNNs become less accurate as the network gets deeper and the input region gets larger. It means that for a large input region, the output intervals of hidden/output neurons computed by symbolic interval analysis for DNNs can be very large. However, the output intervals of their quantized counterparts are always limited by the quantization grid limit, i.e., $[0, \frac{2^Q-1}{2^{Q-2}}]$. Hence, the difference intervals computed in Table 2 can be very conservative for large input regions and deeper networks.

5.2 Effectiveness and Efficiency of QEBVerif

We evaluate QEBVerif on QNNs Ax-z, Py-z for $x = 1$, $y \in \{1, 2, 3, 4\}$ and $z \in \{4, 6, 8, 10\}$, as well as DNNs correspondingly. We use the same input regions and error bounds as in Sect. 5.1 except that we consider $r \in \{3, 6, 13\}$ for each input point for ACAS Xu. Note that, we omit the other two radii for ACAS Xu and use medium-sized QNNs for MNIST as our evaluation benchmarks of this experiment for the sake of time and computing resources.

Figure 4 shows the verification results of QEBVerif within 1 h per task, which gives the number of successfully verified tasks with three methods. Note that only the number of successfully proved tasks is given in Fig. 4 for DRA due to its incompleteness. The blue bars show the results using only the symbolic



(a) ACAS Xu: $Q = 4$. (b) ACAS Xu: $Q = 6$. (c) ACAS Xu: $Q = 8$. (d) ACAS Xu: $Q = 10$.

(e) MNIST: $Q = 4$. (f) MNIST: $Q = 6$. (g) MNIST: $Q = 8$. (h) MNIST: $Q = 10$.

Fig. 4. Verification Results of QEBVerif on ACAS Xu and MNIST.

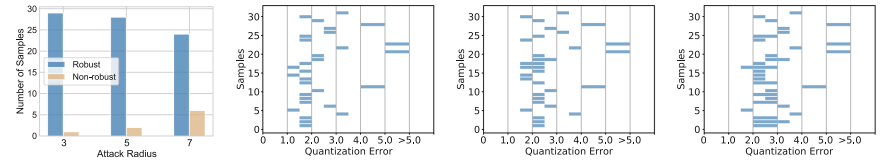
differential reachability analysis, i.e., QEBVerif (Sym). The yellow bars give the results by a full verification process in QEBVerif as shown in Fig. 2, i.e., we first use DRA and then use MILP solving if DRA fails. The red bars are similar to the yellow ones except that linear constraints of the difference intervals of hidden neurons got from DRA are added into the MILP encoding.

Overall, although DRA successfully proved most of the tasks (60.19% with DRA solely), our MILP-based verification method can help further verify many tasks on which DRA fails, namely, 85.67% with DRA+MILP and 88.59% with DRA+MILP+Diff. Interestingly, we find that the effectiveness of the added linear constraints of the difference intervals varies on the MILP solving efficiency on different tasks. Our conjecture is that there are some heuristics in the Gurobi solving algorithm for which the additional constraints may not always be helpful. However, those difference linear constraints allow the MILP-based verification method to verify more tasks, i.e., 79 tasks more in total.

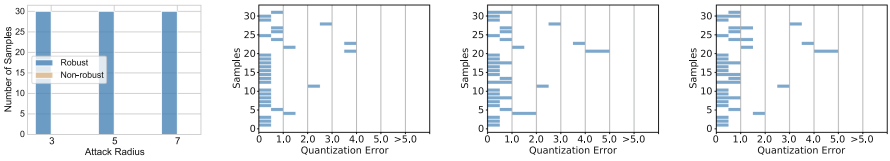
5.3 Correlation of Quantization Errors and Robustness

We use QEBVerif to verify a set of properties $\Psi = \{P(\mathcal{N}, \widehat{\mathcal{N}}, \hat{\mathbf{x}}, r, \epsilon)\}$, where $\mathcal{N}$ = P1-Full, $\widehat{\mathcal{N}} \in \{\text{P1-4, P1-8}\}$, $\hat{\mathbf{x}} \in \mathcal{X}$ and $\mathcal{X}$ is the set of the 30 samples from MNIST as above, $r \in \{3, 5, 7\}$ and $\epsilon \in \Omega = \{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0\}$. We solve all the above tasks and process all the results to obtain the tightest range of quantization error bounds $[a, b]$ for each input region such that $a, b \in \Omega$. It allows us to obtain intervals that are tighter than those obtained via DRA. Finally, we implemented a robustness verifier for QNNs in a way similar to [40] to check the robustness of P1-4 and P1-8 w.r.t. the input regions given in Ψ.

Figure 5 gives the experimental results. The blue (resp. yellow) bars in Figs. 5(a) and 5(e) show the number of robust (resp. non-robust) samples among the 30 verification tasks, and blue bars in the other 6 figures demonstrate the



(a) Robustness Results for P1-4. (b) Errors for P1-4 under $r = 3$. (c) Errors for P1-4 under $r = 5$. (d) Errors for P1-4 under $r = 7$.



(e) Robustness Results for P1-8. (f) Errors for P1-8 under $r = 3$. (g) Errors for P1-8 under $r = 5$. (h) Errors for P1-8 under $r = 7$.

Fig. 5. Distribution of (non-)robust samples and Quantization Errors under radius r and quantization bits Q.

quantization error interval for each input region. By comparing the results of P1-8 and P1-4, we observe that P1-8 is more robust than P1-4 w.r.t. the 90 input regions and its quantization errors are also generally much smaller than that of P1-4. Furthermore, we find that P1-8 remains consistently robust as the radius increases, and its quantization error interval changes very little. However, P1-4 becomes increasingly less robust as the radius increases and its quantization error also increases significantly. Thus, we speculate that there may be some correlation between network robustness and quantization error in QNNs. Specifically, as the quantization bit size decreases, the quantization error increases and the QNN becomes less robust. The reason we suspect "the fewer bits, the less robust" is that with fewer bits, a perturbation may easily cause significant change on hidden neurons (i.e., the change is magnified by the loss of precision) and consequently the output. Furthermore, the correlation between the quantization error bound and the empirical robustness of the QNN suggests that it is indeed possible to apply our method to compute the quantization error bound and use it as a guide for identifying the best quantization scheme which balances the size of the model and its robustness.

6 Related Work

While there is a large and growing body of work on quality assurance techniques for neural networks including testing (e.g., [4–7,47,50,56,57,63,69]) and formal verification (e.g., [2,8,12,13,15,19,24,29,30,32,34,37,38,51,54,55,58–60,62,70]). Testing techniques are often effective in finding violations, but they cannot prove their absence. While formal verification can prove their absence, existing methods typically target real-valued neural networks, i.e., DNNs, and

are not effective in verifying quantization error bound [48]. In this section, we mainly discuss the existing verification techniques for QNNs.

Early work on formal verification of QNNs typically focuses on 1-bit quantized neural networks (i.e., BNNs) [3,9,46,52,53,66,67]. Narodytska et al. [46] first proposed to reduce the verification problem of BNNs to a satisfiability problem of a Boolean formula or an integer linear programming problem. Baluta et al. [3] proposed a PAC-style quantitative analysis framework for BNNs via approximate SAT model-counting solvers. Shih et al. proposed a quantitative verification framework for BNNs [52,53] via a BDD learning-based method [45]. Zhang et al. [66,67] proposed a BDD-based verification framework for BNNs, which exploits the internal structure of the BNNs to construct BDD models instead of BDD-learning. Giacobbe et al. [16] pushed this direction further by introducing the first formal verification for multiple-bit quantized DNNs (i.e., QNNs) by encoding the robustness verification problem into an SMT formula based on the first-order theory of quantifier-free bit-vector. Later, Henzinger et al. [22] explored several heuristics to improve the efficiency and scalability of [16]. Very recently, [40,68] proposed an ILP-based method and an MILP-based verification method for QNNs, respectively, and both outperform the SMT-based verification approach [22]. Though these works can directly verify QNNs or BNNs, they cannot verify quantization error bounds.

There are also some works focusing on exploring the properties of two neural networks which are most closely related to our work. Paulsen et al. [48,49] proposed differential verification methods to verify two DNNs with the same network topology. This idea has been extended to handle recurrent neural networks [41]. The difference between [41,48,49] and our work has been discussed throughout this work, i.e., they focus on quantized weights and cannot handle quantized activation tensors. Moreover, their methods are not complete, thus would fail to prove tighter error bounds. Semi-definite programming was used to analyze the different behaviors of DNNs and *fully* QNNs [33]. Different from our work focusing on verification, they aim at generating an upper bound for the worst-case error induced by quantization. Furthermore, [33] only scales tiny QNNs, e.g., 1 input neuron, 1 output neuron, and 10 neurons per hidden layer (up to 4 hidden layers). In comparison, our differential reachability analysis scales to much larger QNNs, e.g., QNN with 4890 neurons.

7 Conclusion

In this work, we proposed a novel quantization error bound verification method QEBVerif which is sound, complete, and arguably efficient. We implemented it as an end-to-end tool and conducted thorough experiments on various QNNs with different quantization bit sizes. Experimental results showed the effectiveness and the efficiency of QEBVerif. We also investigated the potential correlation between robustness and quantization errors for QNNs and found that as the quantization error increases the QNN might become less robust. For further work, it would be interesting to investigate the verification method for other activation functions and network architectures, towards which this work makes a significant step.

Acknowledgements. This work is supported by the National Key Research Program (2020AAA0107800), National Natural Science Foundation of China (62072309), CAS Project for Young Scientists in Basic Research (YSBR-040), ISCAS New Cultivation Project (ISCAS-PYFX-202201), and the Ministry of Education, Singapore under its Academic Research Fund Tier 3 (MOET32020-0004). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the Ministry of Education, Singapore.

References

1. Amir, G., Wu, H., Barrett, C.W., Katz, G.: An SMT-based approach for verifying binarized neural networks. In: Proceedings of the 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 203–222 (2021)
2. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri, S.: Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 731–744 (2019)
3. Baluta, T., Shen, S., Shinde, S., Meel, K.S., Saxena, P.: Quantitative verification of neural networks and its security applications. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1249–1264 (2019)
4. Bu, L., Zhao, Z., Duan, Y., Song, F.: Taking care of the discretization problem: a comprehensive study of the discretization problem and a black-box adversarial attack in discrete integer domain. IEEE Trans. Dependable Secur. Comput. **19**(5), 3200–3217 (2022)
5. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: Proceedings of the 2017 IEEE Symposium on Security and Privacy, pp. 39–57 (2017)
6. Chen, G., et al.: Who is real Bob? Adversarial attacks on speaker recognition systems. In: Proceedings of the 42nd IEEE Symposium on Security and Privacy, pp. 694–711 (2021)
7. Chen, G., Zhao, Z., Song, F., Chen, S., Fan, L., Liu, Y.: AS2T: arbitrary source-to-target adversarial attack on speaker recognition systems. IEEE Trans. Dependable Secur. Comput., 1–17 (2022)
8. Chen, G., et al.: Towards understanding and mitigating audio adversarial examples for speaker recognition. IEEE Trans. Dependable Secur. Comput., 1–17 (2022)
9. Choi, A., Shi, W., Shih, A., Darwiche, A.: Compiling neural networks into tractable Boolean circuits. In: Proceedings of the AAAI Spring Symposium on Verification of Neural Networks (2019)
10. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, pp. 238–252 (1977)
11. Duncan, K., Komendantskaya, E., Stewart, R., Lones, M.: Relative robustness of quantized neural networks against adversarial attacks. In: Proceedings of the International Joint Conference on Neural Networks, pp. 1–8 (2020)
12. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis, pp. 269–286 (2017)

13. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: Proceedings of the 32nd International Conference on Computer Aided Verification, pp. 43–65 (2020)
14. Eykholt, K., et al.: Robust physical-world attacks on deep learning visual classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1625–1634 (2018)
15. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI^2: safety and robustness certification of neural networks with abstract interpretation. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 3–18 (2018)
16. Giacobbe, M., Henzinger, T.A., Lechner, M.: How many bits does it take to quantize your neural network? In: TACAS 2020. LNCS, vol. 12079, pp. 79–97. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45237-7_5
17. Gong, R., et al.: Differentiable soft quantization: bridging full-precision and low-bit neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 4851–4860 (2019)
18. Google: Tensorflow lite (2022). https://www.tensorflow.org/lite
19. Guo, X., Wan, W., Zhang, Z., Zhang, M., Song, F., Wen, X.: Eager falsification for accelerating robustness verification of deep neural networks. In: Proceedings of the 32nd IEEE International Symposium on Software Reliability Engineering, pp. 345–356 (2021)
20. Gurobi: a most powerful mathematical optimization solver (2018). https://www.gurobi.com/
21. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural network with pruning, trained quantization and Huffman coding. In: Proceedings of the 4th International Conference on Learning Representations (2016)
22. Henzinger, T.A., Lechner, M., Zikelic, D.: Scalable verification of quantized neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 3787–3795 (2021)
23. Hinton, G., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Sig. Process. Mag. **29**(6), 82–97 (2012)
24. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Proceedings of the 29th International Conference on Computer Aided Verification, pp. 3–29 (2017)
25. Jacob, B., et al.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2704–2713 (2018)
26. Julian, K.D., Kochenderfer, M.J., Owen, M.P.: Deep neural network compression for aircraft collision avoidance systems. J. Guid. Control. Dyn. **42**(3), 598–608 (2019)
27. Jung, S., et al.: Learning to quantize deep networks by optimizing quantization intervals with task loss. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4350–4359 (2019)
28. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Li, F.: Large-scale video classification with convolutional neural networks. In: Proceedings of 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1725–1732 (2014)
29. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Proceedings of the 29th International Conference on Computer Aided Verification, pp. 97–117 (2017)

30. Katz, G., et al.: The marabou framework for verification and analysis of deep neural networks. In: Proceedings of the 31st International Conference on Computer Aided Verification, pp. 443–452 (2019)
31. LeCun, Y., Cortes, C.: MNIST handwritten digit database (2010)
32. Li, J., Liu, J., Yang, P., Chen, L., Huang, X., Zhang, L.: Analyzing deep neural networks with symbolic propagation: towards higher precision and faster verification. In: Chang, B.-Y.E. (ed.) SAS 2019. LNCS, vol. 11822, pp. 296–319. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32304-2_15
33. Li, J., Drummond, R., Duncan, S.R.: Robust error bounds for quantised and pruned neural networks. In: Proceedings of the 3rd Annual Conference on Learning for Dynamics and Control, pp. 361–372 (2021)
34. Li, R., et al.: Prodeep: a platform for robustness verification of deep neural networks. In: Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1630–1634 (2020)
35. Lin, D.D., Talathi, S.S., Annapureddy, V.S.: Fixed point quantization of deep convolutional networks. In: Proceedings of the 33nd International Conference on Machine Learning, pp. 2849–2858 (2016)
36. Lin, J., Gan, C., Han, S.: Defensive quantization: when efficiency meets robustness. In: Proceedings of the International Conference on Learning Representations (2019)
37. Liu, J., Xing, Y., Shi, X., Song, F., Xu, Z., Ming, Z.: Abstraction and refinement: towards scalable and exact verification of neural networks. CoRR abs/2207.00759 (2022)
38. Liu, W., Song, F., Zhang, T., Wang, J.: Verifying ReLU neural networks from a model checking perspective. J. Comput. Sci. Technol. **35**(6), 1365–1381 (2020)
39. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward ReLU neural networks. CoRR abs/1706.07351 (2017)
40. Mistry, S., Saha, I., Biswas, S.: An MILP encoding for efficient verification of quantized deep neural networks. IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst. (Early Access) (2022)
41. Mohammadinejad, S., Paulsen, B., Deshmukh, J.V., Wang, C.: DiffRNN: differential verification of recurrent neural networks. In: Proceedings of the 19th International Conference on Formal Modeling and Analysis of Timed Systems, pp. 117–134 (2021)
42. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to Interval Analysis, vol. 110. SIAM (2009)
43. Nagel, M., Amjad, R.A., Van Baalen, M., Louizos, C., Blankevoort, T.: Up or down? Adaptive rounding for post-training quantization. In: Proceedings of the International Conference on Machine Learning, pp. 7197–7206 (2020)
44. Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., van Baalen, M., Blankevoort, T.: A white paper on neural network quantization. arXiv preprint arXiv:2106.08295 (2021)
45. Nakamura, A.: An efficient query learning algorithm for ordered binary decision diagrams. Inf. Comput. **201**(2), 178–198 (2005)
46. Narodytska, N., Kasiviswanathan, S.P., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 6615–6624 (2018)
47. Odena, A., Olsson, C., Andersen, D.G., Goodfellow, I.J.: TensorFuzz: debugging neural networks with coverage-guided fuzzing. In: Proceedings of the 36th International Conference on Machine Learning, pp. 4901–4911 (2019)

48. Paulsen, B., Wang, J., Wang, C.: ReluDiff: differential verification of deep neural networks. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), pp. 714–726. IEEE (2020)
49. Paulsen, B., Wang, J., Wang, J., Wang, C.: NeuroDiff: scalable differential verification of neural networks using fine-grained approximation. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, pp. 784–796 (2020)
50. Pei, K., Cao, Y., Yang, J., Jana, S.: DeepXplore: automated whitebox testing of deep learning systems. In: Proceedings of the 26th Symposium on Operating Systems Principles, pp. 1–18 (2017)
51. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Proceedings of the 22nd International Conference on Computer Aided Verification, pp. 243–257 (2010)
52. Shih, A., Darwiche, A., Choi, A.: Verifying binarized neural networks by Angluin-style learning. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 354–370. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_25
53. Shih, A., Darwiche, A., Choi, A.: Verifying binarized neural networks by local automaton learning. In: Proceedings of the AAAI Spring Symposium on Verification of Neural Networks (2019)
54. Singh, G., Ganvir, R., Püschel, M., Vechev, M.T.: Beyond the single neuron convex barrier for neural network certification. In: Proceedings of the Annual Conference on Neural Information Processing Systems, pp. 15072–15083 (2019)
55. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. (POPL) **3**, 41:1–41:30 (2019)
56. Song, F., Lei, Y., Chen, S., Fan, L., Liu, Y.: Advanced evasion attacks and mitigations on practical ml-based phishing website classifiers. Int. J. Intell. Syst. **36**(9), 5210–5240 (2021)
57. Tian, Y., Pei, K., Jana, S., Ray, B.: DeepTest: automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of the 40th International Conference on Software Engineering, pp. 303–314 (2018)
58. Tran, H.-D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using ImageStars. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 18–42. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_2
59. Tran, H., et al.: Star-based reachability analysis of deep neural networks. In: Proceedings of the 3rd World Congress on Formal Methods, pp. 670–686 (2019)
60. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: Proceedings of the 27th USENIX Security Symposium, pp. 1599–1614 (2018)
61. WikiChip: FSD chip - tesla. https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip. Accessed 30 Apr 2022
62. Yang, P., et al.: Improving neural network verification through spurious region guided refinement. In: Groote, J.F., Larsen, K.G. (eds.) Proceedings of 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 389–408 (2021)
63. Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine learning testing: survey, landscapes and horizons. IEEE Trans. Software Eng. **48**(2), 1–36 (2022)
64. Zhang, Y., Song, F., Sun, J.: QEBVerif (2023). https://github.com/S3L-official/QEBVerif
65. Zhang, Y., Song, F., Sun, J.: QEBVerif: quantization error bound verification of neural networks. CoRR abs/2212.02781 (2023)

66. Zhang, Y., Zhao, Z., Chen, G., Song, F., Chen, T.: BDD4BNN: a BDD-based quantitative analysis framework for binarized neural networks. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12759, pp. 175–200. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_8
67. Zhang, Y., Zhao, Z., Chen, G., Song, F., Chen, T.: Precise quantitative analysis of binarized neural networks: a BDD-based approach. ACM Trans. Software Eng. Methodol. **32**(3) (2023)
68. Zhang, Y., et al.: QVIP: an ILP-based formal verification approach for quantized neural networks. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, pp. 82:1–82:13 (2023)
69. Zhao, Z., Chen, G., Wang, J., Yang, Y., Song, F., Sun, J.: Attack as defense: characterizing adversarial examples using robustness. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 42–55 (2021)
70. Zhao, Z., Zhang, Y., Chen, G., Song, F., Chen, T., Liu, J.: CLEVEREST: accelerating CEGAR-based neural network verification via adversarial attacks. In: Singh, G., Urban, C. (eds.) Proceedings of the 29th International Symposium on Static Analysis, pp. 449–473 (2022). https://doi.org/10.1007/978-3-031-22308-2_20