QEBVerif: Quantization Error Bound Verification of Neural Networks

Yedi Zhang
ShanghaiTech University
Shanghai, China

Fu Song
ShanghaiTech University
Shanghai, China

Jun Sun
Singapore Management University
Singapore

***Abstract*—While deep neural networks (DNNs) have demonstrated impressive performance in solving many challenging tasks, they are limited to resource-constrained devices owing to their demand for computation power and storage space. Quantization is one of the most promising techniques to address this issue by quantizing the weights and/or activation tensors of a DNN into lower bit-width fixed-point numbers. While quantization has been empirically shown to introduce minor accuracy loss, it lacks formal guarantees on that, especially when the resulting quantized neural networks (QNNs) are deployed in safety-critical applications. A majority of existing verification methods focus exclusively on individual neural networks, either DNNs or QNNs. While promising attempts have been made to verify the quantization error bound between DNNs and their quantized counterparts, they are not complete and more importantly do not support *fully* quantified neural networks, namely, only weights are quantized. To fill this gap, in this work, we propose a quantization error bound verification method (QEBVerif), where both weights and activation tensors are quantized. QEBVerif consists of two analyses: 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 efficiently compute a tight quantization error interval. If it 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 arguably efficient. We implement QEBVerif in a tool and conduct extensive experiments, showing its effectiveness and efficiency.**

I. Introduction

In the past few years, the study of deep neural networks (DNNs) has grown at an impressive pace owing to their outstanding performance in solving various complicated tasks [1]–[3]. Modern DNNs are often large in size and contain a great number of parameters which are usually stored as 32/64-bit floating-point numbers. Thus, they often result in high computational cost and excessive storage requirement. With the rise of mobile devices and the Internet of Things, there is an increasing demands for small, energy-efficient neural networks that can be deployed on resource-constrained embedded devices. To address this issue, a number of network compression techniques [4]–[7] have been proposed by removing redundant neurons and quantizing weights and/or activation tensors into lower bit-width fixed-point numbers, so they can drastically reduce memory storage and execution time with bit-wise and/or integer computations while avoiding computational-expensive floating-point computations.

In spite of the empirically impressive results which show there are only minor accuracy loss, these compression techniques often do not preserve properties such as robustness [8]. On the one hand, the main idea of most neural network quantization methods is to minimize the expected impact of quantization on the final task loss and formulate it as an optimization problem [9]–[11]. However, they cannot guarantee that the final output error is always lower than a given error bound, especially when some specified input regions are interested. This is concerning as such errors may lead to catastrophes when the quantized neural networks (QNNs) are deployed in safety-critical applications [12], [13]. On the other hand, analyzing (in particular, quantifying) such errors can also help us understand how compression techniques such as network pruning and quantization affect the network behaviors [14], and provide insights on, for instance, how to choose appropriate quantization bit sizes without introducing too much errors. Therefore, a method which formally verifies and quantifies the errors between DNNs and their quantized counterparts is highly desirable.

Thanks to the long-standing line of verification and validation (V&V) research in software engineering, there is a large and growing body of work on developing V&V techniques for DNNs [15]–[23], aiming to establish a formal guarantee on the network behaviors. Some V&V methods for QNNs [8], [24]–[28] have been also proposed. However, all the above-mentioned methods focus exclusively on verifying individual neural networks that are either DNNs or QNNs. While an existing individual DNN verification tool could be adapted to verify quantization error bound by combining a DNN and its quantized counterpart into one DNN, in practice, the bounds computed by this approach with state-of-the-art DNN verification tools are too loose [29], [30]. Therefore, Paulsen et al. [29], [30] proposed differential verification methods which gives formal guarantees on the difference bounds of two DNNs with the same network topology. 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. Recently, this idea has been extended to handle recurrent neural networks

(RNNs) [31] though the refinement is not considered. While their methods [29]–[31] can be used to analyze the error bound introduced by quantization when the weights of a DNN are quantized into lower bit-width fixed-point numbers, they are not complete due to the propagation of approximation error, thus would not be able to prove tighter error bounds. More importantly, they cannot handle the practical cases where both the weights and the activation tensors of the compressed network are quantized to lower bit-width fixed-point numbers.

To fill this gap, in this work, we propose a sound and complete **Q**uantization **E**rror **B**ound **Verif**ication method (QEBVerif) to efficiently and effectively verify if the quantization errors 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 [14]), where fully means that both of the weights and activation tensors of the DNN are quantized into lower bit-width fixed-point numbers. To achieve this goal, there are two main technical challenges. First, the activation tensors in a QNN are discrete values rather than continuous values, which contribute additional rounding errors to the final quantization errors. Such rounding errors introduced by hidden neurons in the QNN are hard to propagate symbolically, which makes it more difficult to give relatively accurate difference intervals. Second, there are much more activation patterns, i.e., $3 \times 6 = 18$ patterns (3 for DNNs and 6 for QNNs), to consider for the difference intervals in a forward propagation, while in [29], [30] there are only 9 patterns. How to handle these different patterns efficiently and soundly is highly nontrivial.

To tackle the above challenges, QEBVerif first conducts a reachability analysis to quantify the quantization errors, which is called *differential reachability analysis* (DRA) in this work. Such an analysis has 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*, then 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.

The DRA of QEBVerif performs sound transformations for the affine and activation functions to propagate quantization errors of two networks layer-by-layer. Furthermore, for the affine transformation, QEBVerif provides two alternative solutions: (1) *concrete-based*, and (2) *symbolic-based*. The former directly computes sound and concrete difference intervals via interval analysis [32], while the latter leverages abstract interpretation [33] to compute sound and symbolic difference intervals based on an abstract domain for QNNs. In comparison, the symbolic-based one is more accurate but less efficient than the concrete-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 a naive approach is often too conservative as it induces a significant loss of accuracy.

To resolve the problem that cannot be proved via DRA, we resort the sound and complete MILP-based verification method. Inspired by the MILP encoding of DNN verification [34], 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 DRA as mixed-integer linear constraints to boost the verification.

We implement our method as an end-to-end tool and use Gurobi [35] as our back-end MILP solver. We extensively evaluate it on a large set of verification tasks, where the number of neurons varies from 894 to 1818, 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 our difference propagation approaches with a baseline that first independently computes the output intervals of DNNs and QNNs using the existing state-of-the-art symbolic interval analysis [36] and concrete interval analysis [28] respectively, and then conducts an interval subtraction. The experimental results show that both our concrete- 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. Finally, we study the potential correlation of quantization errors and robustness for QNNs using QEBVerif.

We summarize our contributions as follows:

- We introduce a sound, complete and efficient quantization error bound verification method QEBVerif by combining a DRA and an MILP-based verification method.
- We propose a novel DRA to compute sound and tight quantization error intervals accompanied by an abstract domain for QNNs, which can significantly and soundly tighten the quantization error intervals.
- We implement QEBVerif as an end-to-end tool and empirically evaluate QEBVerif on a large set of verification tasks and demonstrate its effectiveness and efficiency.

Outline. Section II introduces the preliminaries of this work. Section III presents our quantization error bound verification method QEBVerif and Section IV gives our differential reachability analysis. Section V reports experimental results. We discuss related work in Section VI. Finally, we conclude this work in Section VII.

II. 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 $\{1, \dots, n\}$ 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. Given a matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, $\mathbf{W}_{i,j}$ denote the j-entry in the i-th row of the matrix $\mathbf{W}$. $\mathbf{x}_i$ denotes the i-th entry of the vector $\mathbf{x}$.

A. Deep 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 are pointed to by only the neurons in the preceding layer.

Definition 1 (Deep Neural Network). *A 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 $2 \le i \le d-1$ 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 $l_d : \mathbb{R}^{d-1} \to \mathbb{R}^s$ 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 the most commonly used activation functions: the rectified linear unit (ReLU) function, defined as $\text{ReLU}(x) = \max(x, 0)$.

Example 1. *An example of DNN $\mathcal{N}_e$ with 3 layers (one input layer, one hidden layer, and one output layer) is given Fig. 1(a). The entries of the weight matrices are associated with the edges, and all the biases are 0. Given an input $\mathbf{x} = (0.78, 0.45)$, the outputs of the input, hidden and output layers are $\mathbf{x}^1 = (0.78, 0.45)$, $\mathbf{x}^2 = (0.7869, 0)$ and $\mathbf{x}^3 = (0.220332)$, respectively.*

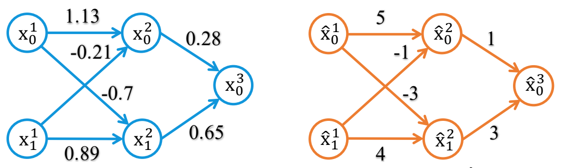
B. Quantized Neural Networks

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 [37]. 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:

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



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

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

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:

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

For example, given a quantization configuration $\langle +, 4, 2 \rangle$, we can transfer the real value $x = 1.51$ to its integer counterpart $\hat{x} = \lfloor 1.51 \cdot 2^2 \rceil = \lfloor 6.04 \rceil = 6$ and fixed-point counterpart $\tilde{x} = 6/2^2 = 1.5$. We next introduce how the quantization carries out in each layer of a DNN to get its QNN w.r.t. the symmetric uniform quantization scheme. Note that, 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.

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 $2 \le i \le d-1$ is the function such that for each $j \in [n_i]$,*
$$\hat{\mathbf{x}}_j^i = \text{clamp}(\lfloor 2^{F_i} \widehat{\mathbf{W}}_{j,:}^i \hat{\mathbf{x}}^{i-1} + 2^{F_h - F_b} \hat{\mathbf{b}}_j^i \rceil,\ \mathcal{C}_h^{\text{lb}},\ \mathcal{C}_h^{\text{ub}}),$$
where F_i is $F_h - F_w - F_{in}$ if $i = 2$, and $-F_w$ otherwise; and $\widehat{\mathbf{W}}_{j,k}^i = \text{clamp}(\lfloor 2^{F_w} \mathbf{W}_{j,k}^i \rceil, \mathcal{C}_w^{\text{lb}}, \mathcal{C}_w^{\text{ub}})$ and $\hat{\mathbf{b}}_j^i = \text{clamp}(\lfloor 2^{F_b} \mathbf{b}_j^i \rceil, \mathcal{C}_b^{\text{lb}}, \mathcal{C}_b^{\text{ub}})$ for each $k \in [n_{i-1}]$.
- *Output layer $\hat{l}_d : \mathbb{Z}^{d-1} \to \mathbb{R}^s$ is the function such that $\hat{\mathbf{y}} = \hat{\mathbf{x}}^d = \hat{l}_d(\hat{\mathbf{x}}^{d-1}) = 2^{-F_w} \widehat{\mathbf{W}}^d \hat{\mathbf{x}}^{d-1} + 2^{F_h - F_b} \hat{\mathbf{b}}^d$*

Example 2. *Consider the DNN $\mathcal{N}_e$ given in Example 1. The quantization configurations for the weights, 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(b) where the quantized weights are associated with the edges. Given the quantized input $\hat{\mathbf{x}} = (12, 7)$, the outputs of the input, hidden and output layers are $\hat{\mathbf{x}}^1 = (12, 7)$, $\hat{\mathbf{x}}^2 = (3, 0)$ and $\hat{\mathbf{x}}^3 = (0.75)$, respectively.*

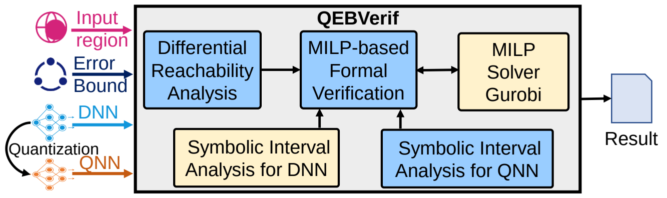

Input region
Error Bound
DNN
Quantization
QNN
QEBVerif
Differential Reachability Analysis
MILP-based Formal Verification
MILP Solver Gurobi
Symbolic Interval Analysis for DNN
Symbolic Interval Analysis for QNN
Result

Fig. 2: An overview of QEBVerif.

C. Quantization Error Bound Verification

We now formulate the quantization error bound verification problem considered in this work.

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* $R(\hat{\mathbf{x}}, r) = \{\hat{x}' \in \mathbb{Z}^n \mid ||\hat{\mathbf{x}}' - \hat{\mathbf{x}}||_\infty \leq r\}$ *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* $\mathbf{x}' = \hat{\mathbf{x}}'/\mathcal{C}_{in}^{\text{ub}}$.

Note that, we get the input for DNN via dividing $\hat{\mathbf{x}}'$ by $\mathcal{C}_{in}^{\text{ub}}$ to allow input normalization. Furthermore, 2^{-F_h} is used to align the precision between the outputs of QNN and DNN.

Example 3. *Consider the DNN* $\mathcal{N}_e$ *and corresponding QNN* $\widehat{\mathcal{N}}_e$ *given in Fig. 1(a) and Fig. 1(b). Given a quantized input* $\hat{\mathbf{x}} = (12, 7)$ *and a radius* $r = 0$*, the input region for QNN* $\widehat{\mathcal{N}}_e$ *is* $R((12, 7), 0) = \{(12, 7)\}$*. Since* $\mathcal{C}_{in}^{\text{ub}} = 15$*, by Definition 3, we have* $||2^{-2}\widehat{\mathcal{N}}_e((12, 7)) - \mathcal{N}(12/15, 7/15)||_\infty = |0.1875 - 0.22568| = 0.03818$*. Then,* $\widehat{\mathcal{N}}_e$ *has a quantization error bound of* ϵ *w.r.t. the input region* $R((12, 7), 0)$ *for any* $\epsilon > 0.03818$*.*

One may also encode the quantization error bound property $P(\mathcal{N}, \widehat{\mathcal{N}}, \hat{\mathbf{x}}, r, \epsilon)$ as follows:

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

Then, the quantization error bound verification problem is to check if the property $P(\mathcal{N}, \widehat{\mathcal{N}}, \hat{\mathbf{x}}, r, \epsilon)$ holds. In other words, it checks whether the output difference between the QNN $\widehat{\mathcal{N}}$ and the DNN $\mathcal{N}$ for all the similar inputs (according to the norm) is always smaller than a given threshold ϵ.

Specifically, for a classification task, we only focus on the output difference of the predicted class instead of all the classes. Given a DNN $\mathcal{N}$, a corresponding QNN $\widehat{\mathcal{N}}$ and a quantized input $\hat{\mathbf{x}}$ which is classified to class g by the DNN $\mathcal{N}$, then the quantization error bound property $P^c(\mathcal{N}, \widehat{\mathcal{N}}, \hat{\mathbf{x}}, r, \epsilon)$ for a classification task is as follows:

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

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

In this work, we focus on the quantization error bound verification problem for classification tasks.

III. Methodology of QEBVerif

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

A. 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 $P^c(\mathcal{N}, \widehat{\mathcal{N}}, \hat{\mathbf{x}}, r, \epsilon)$, QEBVerif first performs a differential reachability analysis (DRA) to compute a sound output difference interval for the two networks during which 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 formal verification method by encoding the problem into an equivalent mixed integer linear programming (MILP) problem which can be solved by the 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 without loss of accuracy. 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 will increase 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 arguably 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.

B. 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 a way is often ineffective due to the significant precision loss.

Recently, Paulsen et al. [29] 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), 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 been improved to tighten the approximated difference intervals [30] and extended to recurrent neural networks [31]. However, as mentioned previously, they do not support *fully* quantified neural networks. Inspired by them, we design the 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 input 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

Algorithm 1: Forward Difference Propagation

```
Input : DNN $\mathcal{N}$, QNN $\widehat{\mathcal{N}}$, input region $R(\hat{x}, r)$
output: Output difference interval $\delta$
1  Compute $S^{in}(\mathbf{x}^i_j)$, $S(\mathbf{x}^i_j)$, $S^{in}(\hat{\mathbf{x}}^i_j)$ and $S(\hat{\mathbf{x}}^i_j)$ for $i \in [d-1]$,
     $j \in [n_i]$ by applying (symbolic) interval analysis [28], [36];
2  Initialize the difference: $\delta_1 = (2^{-F_{in}} - 1/\mathcal{C}^{\text{ub}}_{in})S(\hat{\mathbf{x}}^1)$;
3  for $i$ in $2, \dots, d-1$ do  // propagate in hidden layers
4      for $j$ in $1, \dots, n_i$ do
5          $\Delta\mathbf{b}^i_j = 2^{-F_b}\hat{\mathbf{b}}^i_j - \mathbf{b}^i_j$;
6          $\xi = 2^{-F_h - 1}$;
7          $\delta^{in}_{i,j} = \text{AffTrs}(\mathbf{W}^i_{j,:}, 2^{-F_w}\widehat{\mathbf{W}}^i_{j,:}, \Delta\mathbf{b}^i_j, S(\mathbf{x}^{i-1}), \delta_{i-1}, \xi)$;
8          $\delta_{i,j} = \text{ActTrs}(\delta^{in}_{i,j}, S^{in}(\mathbf{x}^i_j), 2^{-F_h}S^{in}(\hat{\mathbf{x}}^i_j))$;
9  // propagate in the output layer
10 for $j$ in $1, \dots, n_d$ do
11     $\Delta\mathbf{b}^d_j = 2^{-F_b}\hat{\mathbf{b}}^d_j - \mathbf{b}^d_j$;
12     $\delta_{d,j} = \delta^{in}_{d,j} = \text{AffTrs}(\mathbf{W}^d_{j,:}, 2^{-F_w}\widehat{\mathbf{W}}^d_{j,:}, \Delta\mathbf{b}^d_j, S(\mathbf{x}^{d-1}), \delta_{d-1}, 0)$;
13 return $(\delta_{i,j})_{2\le i\le d, 1\le j\le n_d}$
```

activation functions, and use $\delta^{in}_{i,j}$ (resp. $\delta_{i,j}$) to denote the interval for the j-th neuron of that. LB$(\cdot)$ (resp. UB$(\cdot)$) denotes the concrete lower bound (resp. upper bound) accordingly.

Based on these notations, we formulate 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 the intervals $S^{in}(\mathbf{x}^i_j)$ and $S(\mathbf{x}^i_j)$ for each neuron in $\mathcal{N}$ using DeepPoly [36], and compute the intervals $S^{in}(\hat{\mathbf{x}}^i_j)$ and $S(\hat{\mathbf{x}}^i_j)$ for each neuron in $\widehat{\mathcal{N}}$ using interval analysis [28]. By Definition 3, we can get the output difference of input layer as $\delta_1 = (2^{-F_{in}} - 1/\mathcal{C}^{\text{ub}}_{in})S(\hat{\mathbf{x}}^1)$. Then, we compute the output difference δ_i of each hidden layer iteratively by applying the affine transformer and activation transformer given in Algorithms 2 and 3. Finally, we get the output difference δ_d for the output layer using the affine transformer.

1) Affine Transformer: The difference before applying the activation function for the j-th neuron in the i-th layer is: $\delta^{in}_{i,j} = 2^{-F_h}\lfloor 2^{F_i}\widehat{\mathbf{W}}^i_{j,:}S(\hat{\mathbf{x}}^{i-1}_k) + 2^{F_h - F_b}\hat{\mathbf{b}}^i_j \rceil - \mathbf{W}^i_{j,:}S(\mathbf{x}^{i-1}) - \mathbf{b}^i_j$ where 2^{-F_h} is used to align the precision between the outputs of the two networks (cf. Section II).

By replacing $2^{F_i}\widehat{\mathbf{W}}^i_{j,:}S(\hat{\mathbf{x}}^{i-1}) + 2^{F_h - F_b}\hat{\mathbf{b}}^i_j$ with $S(\mathbf{z}^i_j)$, we have $\delta^{in}_{i,j} = 2^{-F_h}\lfloor S(\mathbf{z}^i_j)\rceil - \mathbf{W}^i_{j,:}S(\mathbf{x}^{i-1}) - \mathbf{b}^i_j$. Then, we can soundly remove the rounding function and give constraints for upper bounds as well as lower bounds of $\delta^{in}_{i,j}$ as follows:

$$\text{UB}(\delta^{in}_{i,j}) \le \text{UB}(2^{-F_h}(S(\mathbf{z}^i_j) + 0.5) - \mathbf{W}^i_{j,:}S(\mathbf{x}^{i-1}) - \mathbf{b}^i)$$

$$\text{LB}(\delta^{in}_{i,j}) \ge \text{LB}(2^{-F_h}(S(\mathbf{z}^i_j) - 0.5) - \mathbf{W}^i_{j,:}S(\mathbf{x}^{i-1}) - \mathbf{b}^i)$$

By putting $S(\mathbf{z}^i_j) = 2^{F_i}\widehat{\mathbf{W}}^i_{j,:}S(\hat{\mathbf{x}}^{i-1}) + 2^{F_h - F_b}\hat{\mathbf{b}}^i_j$ back into the two inequalities above, we have:

$$\begin{aligned}\text{UB}(\delta^{in}_{i,j}) &\le \text{UB}\big(\widetilde{\mathbf{W}}^i_{j,:}S(\tilde{\mathbf{x}}^{i-1}) - \mathbf{W}^i_{j,:}S(\mathbf{x}^{i-1})\big) + \Delta\mathbf{b}^i_j + \xi \\ &= \text{UB}\big(\widetilde{\mathbf{W}}^i_{j,:}\delta_{i-1} + \Delta\mathbf{W}^i_{j,:}S(\mathbf{x}^{i-1})\big) + \Delta\mathbf{b}^i_j + \xi\end{aligned}$$

$$\begin{aligned}\text{LB}(\delta^{in}_{i,j}) &\ge \text{LB}\big(\widetilde{\mathbf{W}}^i_{j,:}S(\tilde{\mathbf{x}}^{i-1}) - \mathbf{W}^i_{j,:}S(\mathbf{x}^{i-1})\big) + \Delta\mathbf{b}^i_j - \xi \\ &= \text{LB}\big(\widetilde{\mathbf{W}}^i_{j,:}\delta_{i-1} + \Delta\mathbf{W}^i_{j,:}S(\mathbf{x}^{i-1})\big) + \Delta\mathbf{b}^i_j - \xi\end{aligned}$$

Algorithm 2: AffTrs Function

```
Input : Weight vector $\mathbf{W}^i_{j,:}$, weight vector $\widetilde{\mathbf{W}}^i_{j,:}$, bias difference
        $\Delta\mathbf{b}^i_j$, neuron interval $S(\mathbf{x}^{i-1})$, difference interval $\delta_{i-1}$,
        rounding error $\xi$
output: Difference interval $\delta^{in}_{i,j}$
1  $lb = \text{LB}\big(\widetilde{\mathbf{W}}^i_{j,:}\delta_{i-1} + (\widetilde{\mathbf{W}}^i_{j,:} - \mathbf{W}^i_{j,:})S(\mathbf{x}^{i-1})\big) + \Delta\mathbf{b}^i_j - \xi$
2  $ub = \text{UB}\big(\widetilde{\mathbf{W}}^i_{j,:}\delta_{i-1} + (\widetilde{\mathbf{W}}^i_{j,:} - \mathbf{W}^i_{j,:})S(\mathbf{x}^{i-1})\big) + \Delta\mathbf{b}^i_j + \xi$
3  return $[lb, ub]$
```

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. $\delta_{i-1} = S(\tilde{\mathbf{x}}^{i-1}) - S(\mathbf{x}^{i-1})$. $\widetilde{\mathbf{W}}^i_{j,:} = 2^{-F_w}\widehat{\mathbf{W}}^i_{j,:}$, $\Delta\mathbf{W}^i_{j,:} = \widetilde{\mathbf{W}}^i_{j,:} - \mathbf{W}^i_{j,:}$, $\Delta\mathbf{b}^i_j = 2^{-F_b}\hat{\mathbf{b}}^i_j - \mathbf{b}^i_j$ and $\xi = 2^{-F_h - 1}$.

Proposition 1. *Algorithms 1 and 2 are sound, i.e., the results over-approximate the underlying real output bounds.* □

2) Activation Transformer: Now we formulate 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 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 the activation transformer in Algorithm 3 is sound.*

Recall that we consider the ReLU activation function in this work, thus τ_h should be $+$, i.e., unsigned. We omit the proof for the sake of space, which can be found at our website [38].

In Section IV, we will present a symbolic interval analysis for QNNs which is able to compute tighter intervals than the concrete one [28], and a more accurate affine transformer than Algorithm 2, both based on an abstract domain of QNNs.

C. MILP Encoding of the Verification Problem

If DRA fails to prove the quantization error bound problem, 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 quantized the input region as a set of integer linear constraints. We adopt the MILP encoding of DNNs [34] to transform the DNN into a set of linear constraints. We also propose a novel MILP encoding for QNNs. Unluckily, we found that this encoding is the same as the one published very recently in TCAD (early access) [39], except that we use (symbolic) intervals to further reduce the size of linear constraints similar to [34] while [39] did not. Thus, in this work, we do not claim this contribution and omit the detailed encoding as well as the encoding of the quantized input region which is also provided in [39]. 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.

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

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
    $lb$ = clamp($\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j))$,0, $t$);  $ub$ = 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
        $lb$ = clamp($\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j))$, 0, $t$)$-\mathsf{UB}(S^{in}(\mathbf{x}^i_j))$;
        $ub$ = 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
        $lb$ = clamp($\mathsf{LB}(S^{in}(\tilde{\mathbf{x}}^i_j))$, 0, $t$)$-\mathsf{UB}(S^{in}(\mathbf{x}^i_j))$;
        $ub$ = 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($\mathsf{LB}(\delta^{in}_{i,j})$,$-\mathsf{UB}(S^{in}(\mathbf{x}^i_j))$));
        $ub$ = clamp($\mathsf{UB}(\delta^{in}_{i,j})$, 0, $t$);
return $[lb, ub]$;
```

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 $\hat{\mathbf{x}}^d_g$ (resp. $\mathbf{x}^d_g$) 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 $\eta = \max(2^{-F_h}\hat{\mathbf{x}}^d_g - \mathbf{x}^d_g, 0)$ can be encoded by the set Θ_g of constraints with an extremely large number $\mathbf{M}$:

$$\Theta_g = \left\{ \begin{array}{c} \eta \geq 0, \\ \eta \geq 2^{-F_h}\hat{\mathbf{x}}^d_g - \mathbf{x}^d_g, \\ \eta \leq \mathbf{M} \cdot v, \\ \eta \leq 2^{-F_h}\hat{\mathbf{x}}^d_g - \mathbf{x}^d_g + \mathbf{M} \cdot (1 - v) \end{array} \right\}$$

As a result, $|2^{-F_h}\hat{\mathbf{x}}^d_g - \mathbf{x}^d_g| \geq \epsilon$ iff the set of linear constraints $\Theta_\epsilon = \Theta_g \cup \{2\eta - (2^{-F_h}\hat{\mathbf{x}}^d_g - \mathbf{x}^d_g) \geq \epsilon\}$ 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.

IV. An Abstract Domain for QNNs

While Algorithm 1 can compute difference intervals, the affine transformer explicitly adds a concrete rounding error interval to each neuron, which will be accumulated leading to significant precision loss over the subsequent layers. To alleviate this problem, we introduce an abstract domain based on DeepPoly [36] which helps to compute sound symbolic approximations for the lower and upper bounds of each difference interval, hence computing tighter difference intervals.

A. DeepPoly*: An abstract domain for DNNs*

Each hidden neuron $\mathbf{x}^i_j$ in a DNN is seen as two nodes $\mathbf{x}^i_{j,0}$ and $\mathbf{x}^i_{j,1}$, such that $\mathbf{x}^i_{j,0} = \sum_{k=1}^{n_{i-1}} \mathbf{W}^i_{j,k}\mathbf{x}^{i-1}_{k,1} + \mathbf{b}^i_j$ (affine function) and $\mathbf{x}^i_{j,1} = \text{ReLU}(\mathbf{x}^i_{j,0})$ (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 $\mathbf{x}^{i-1}_{k,1}$'s. 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 $\mathcal{A}^i_{j,s}$ for the node $\mathbf{x}^i_{j,s}$ ($s \in \{0, 1\}$) is a tuple $\mathcal{A}^i_{j,s} = \langle \mathbf{a}^{i,\leq}_{j,s}, \mathbf{a}^{i,\geq}_{j,s}, l^i_{j,s}, u^i_{j,s} \rangle$, where $\mathbf{a}^{i,\leq}_{j,s}$ and $\mathbf{a}^{i,\geq}_{j,s}$ are respectively the lower and upper polyhedral computations in the form of a linear combination of the variables $\mathbf{x}^{i-1}_{k,1}$'s (if $s = 0$) or $\mathbf{x}^i_{k,0}$'s if $s = 1$, $l^i_{j,s} \in \mathbb{R}$ and $u^i_{j,s} \in \mathbb{R}$ are the concrete lower and upper bound of the neuron. Then, the concretization of the abstract element $\mathcal{A}^i_{j,s}$ is

$$\Gamma(\mathcal{A}^i_{j,s}) = \{x \in \mathbb{R} \mid \mathbf{a}^{i,\leq}_{j,s} \leq x \wedge x \leq \mathbf{a}^{i,\geq}_{j,s}\}.$$

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

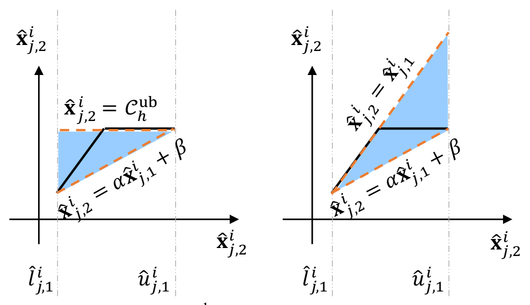
For ReLU function $\mathbf{x}^i_{j,1} = \text{ReLU}(\mathbf{x}^i_{j,0})$, there are three cases to consider of the abstract element $\mathcal{A}^i_{j,1}$:

- 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, all abstract elements in this abstract domain satisfy the following invariant: $\Gamma(\mathcal{A}^i_{j,s}) \subseteq [l^i_{j,s}, u^i_{j,s}]$, which is the key to guarantee soundness.

B. An abstract domain for QNNs

Recall that the activation function in a QNN $\widehat{\mathcal{N}}$ is a min-ReLU function $\min(\text{ReLU}(\lfloor\cdot\rceil), \mathcal{C}^{\text{ub}}_h)$. Thus, we regard each



$\hat{\mathbf{x}}^i_{j,2}$
$\hat{\mathbf{x}}^i_{j,2} = \mathcal{C}^{\text{ub}}_h$
$\hat{\mathbf{x}}^i_{j,2} = \alpha\hat{\mathbf{x}}^i_{j,1} + \beta$
$\hat{\mathbf{x}}^i_{j,2}$
$\hat{l}^i_{j,1}$
$\hat{u}^i_{j,1}$
(a) $(\gamma, \mu) = (0, \mathcal{C}^{\text{ub}}_h)$
$\hat{\mathbf{x}}^i_{j,2}$
$\hat{\mathbf{x}}^i_{j,2} = \hat{\mathbf{x}}^i_{j,1}$
$\hat{\mathbf{x}}^i_{j,2} = \alpha\hat{\mathbf{x}}^i_{j,1} + \beta$
$\hat{\mathbf{x}}^i_{j,2}$
$\hat{l}^i_{j,1}$
$\hat{u}^i_{j,1}$
(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 $\beta = \frac{(\hat{u}^i_{j,1} - \mathcal{C}^{\text{ub}}_h)}{\hat{u}^i_{j,1} - \hat{l}^i_{j,1}}$.

hidden neuron $\hat{\mathbf{x}}^i_j$ in a QNN as three nodes $\hat{\mathbf{x}}^i_{j,0}$, $\hat{\mathbf{x}}^i_{j,1}$, and $\hat{\mathbf{x}}^i_{j,2}$ such that $\hat{\mathbf{x}}^i_{j,0} = \lfloor 2^{F_i} \sum_{k=1}^{n_{i-1}} \widehat{\mathbf{W}}^i_{j,k} \hat{\mathbf{x}}^{i-1}_{k,2} + 2^{F_h - F_b} \hat{\mathbf{b}}^i_j \rceil$ (affine function), $\hat{\mathbf{x}}^i_{j,1} = \max(\hat{\mathbf{x}}^i_{j,0}, 0)$ (ReLU function) and $\hat{\mathbf{x}}^i_{j,2} = \min(\hat{\mathbf{x}}^i_{j,1}, \mathcal{C}^{\text{ub}}_h)$ (min function). We now give the abstract domain $\widehat{\mathcal{A}}^i_{j,p} = \langle \hat{\mathbf{a}}^{i,\leq}_{j,p}, \hat{\mathbf{a}}^{i,\geq}_{j,p}, \hat{l}^i_{j,p}, \hat{u}^i_{j,p} \rangle$ for each neuron $\hat{\mathbf{x}}^i_{j,p}$ ($p \in \{0, 1, 2\}$) in a QNN as follows.

Following DeepPoly, $\hat{\mathbf{a}}^{i,\leq}_{j,0}$ and $\hat{\mathbf{a}}^{i,\geq}_{j,0}$ for the affine function of $\hat{\mathbf{x}}^i_{j,0}$ are defined as

- $\hat{\mathbf{a}}^{i,\leq}_{j,0} = 2^{F_i} \sum_{k=1}^{n_{i-1}} \widehat{\mathbf{W}}^i_{j,k} \hat{\mathbf{x}}^{i-1}_{k,2} + 2^{F_h - F_b} \hat{\mathbf{b}}^i_j - 0.5$ and
- $\hat{\mathbf{a}}^{i,\geq}_{j,0} = 2^{F_i} \sum_{k=1}^{n_{i-1}} \widehat{\mathbf{W}}^i_{j,k} \hat{\mathbf{x}}^{i-1}_{k,2} + 2^{F_h - F_b} \hat{\mathbf{b}}^i_j + 0.5$.

We note that $+0.5$ and -0.5 here are added to soundly encode the rounding function and have no effect on the preservance of invariant. The abstract transformer for the ReLU function $\mathbf{x}^i_{j,1} = \text{ReLU}(\mathbf{x}^i_{j,0})$ is defined the same as DeepPoly.

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

- If $\hat{l}^i_{j,1} \geq \mathcal{C}^{\text{ub}}_h$, then $\hat{\mathbf{a}}^{i,\leq}_{j,2} = \hat{\mathbf{a}}^{i,\geq}_{j,2} = \mathcal{C}^{\text{ub}}_h$, $\hat{l}^i_{j,2} = \hat{u}^i_{j,2} = \mathcal{C}^{\text{ub}}_h$;
- If $\hat{u}^i_{j,1} \leq \mathcal{C}^{\text{ub}}_h$, then $\hat{\mathbf{a}}^{i,\leq}_{j,2} = \hat{\mathbf{a}}^{i,\leq}_{j,1}$, $\hat{\mathbf{a}}^{i,\geq}_{j,2} = \hat{\mathbf{a}}^{i,\geq}_{j,1}$, $\hat{l}^i_{j,2} = \hat{l}^i_{j,1}$ and $\hat{u}^i_{j,2} = \hat{u}^i_{j,1}$;
- 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:* $\Gamma(\widehat{\mathcal{A}}^i_{j,2}) \subseteq [\hat{l}^i_{j,2}, \hat{u}^i_{j,2}]$.

Proof. If $\hat{l}^i_{j,1} \geq \mathcal{C}^{\text{ub}}_h$, then $\hat{\mathbf{a}}^{i,\leq}_{j,2} = \hat{\mathbf{a}}^{i,\geq}_{j,2} = \hat{\mathbf{x}}^i_{j,2} = \mathcal{C}^{\text{ub}}_h$, thus $\mathcal{C}^{\text{ub}}_h = \hat{l}^i_{j,2} = \hat{\mathbf{a}}^{i,\leq}_{j,2} = \hat{\mathbf{x}}^i_{j,2} = \hat{\mathbf{a}}^{i,\geq}_{j,2} = \mathcal{C}^{\text{ub}}_h = \hat{u}^i_{j,2}$. If $\hat{u}^i_{j,1} \leq \mathcal{C}^{\text{ub}}_h$, then $\hat{l}^i_{j,2} = \hat{l}^i_{j,1} \leq \hat{\mathbf{a}}^{i,\leq}_{j,1} = \hat{\mathbf{a}}^{i,\leq}_{j,2} \leq \hat{\mathbf{x}}^i_{j,2} \leq \hat{\mathbf{a}}^{i,\geq}_{j,2} = \hat{\mathbf{a}}^{i,\geq}_{j,1} \leq \hat{u}^i_{j,1} = \hat{u}^i_{j,2}$. Otherwise, we have $\hat{l}^i_{j,1} < \mathcal{C}^{\text{ub}}_h$, $\hat{u}^i_{j,1} > \mathcal{C}^{\text{ub}}_h$, and therefore $\hat{l}^i_{j,2} = \hat{l}^i_{j,1} \leq \hat{\mathbf{x}}^i_{j,2} \leq \lambda \hat{u}^i_{j,1} + \mu = \hat{u}^i_{j,2}$. □

From our abstract domain for QNNs, we get a symbolic interval analysis, similar to the one for DNNs using DeepPoly, to replace the concrete one [28] at Line 1 in Algorithm 1.

C. Symbolic Difference 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, in which all the polyhedral computations are linear combinations of the input variables of the DNN and QNN, respectively, i.e.,
$\mathbf{a}^{i,\leq,*}_{j,s} = \sum_{k=1}^{m} \mathbf{w}^{l,*}_k \mathbf{x}^1_k + \mathbf{b}^{l,*}_j$, $\mathbf{a}^{i,\geq,*}_{j,s} = \sum_{k=1}^{m} \mathbf{w}^{u,*}_k \mathbf{x}^1_k + \mathbf{b}^{u,*}_j$,
$\hat{\mathbf{a}}^{i,\leq,*}_{j,p} = \sum_{k=1}^{m} \hat{\mathbf{w}}^{l,*}_k \hat{\mathbf{x}}^1_k + \hat{\mathbf{b}}^{l,*}_j$, $\hat{\mathbf{a}}^{i,\geq,*}_{j,p} = \sum_{k=1}^{m} \hat{\mathbf{w}}^{u,*}_k \hat{\mathbf{x}}^1_k + \hat{\mathbf{b}}^{u,*}_j$.

The sound lower bound $\Delta l^{i,*}_{j,s}$ and upper $\Delta u^{i,*}_{j,s}$ bound of the difference can be formulated as:

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

where $p = 2s$.

For the lower bound $\Delta l^{i,*}_{j,s}$, we have following derivations:

$$\begin{aligned}
\Delta l^{i,*}_{j,s} &= 2^{-F_h} \hat{\mathbf{a}}^{i,\leq,*}_{j,p} - \mathbf{a}^{i,\geq,*}_{j,s} \\
&= \Delta \mathbf{b}^{l,*}_j + \textstyle\sum_{k=1}^{m} (2^{-F_h} \hat{\mathbf{w}}^{l,*}_k \hat{\mathbf{x}}^1_k - \mathbf{w}^{u,*}_k \mathbf{x}^1_k) \\
&= \Delta \mathbf{b}^{l,*}_j + \textstyle\sum_{k=1}^{m} (\tilde{\mathbf{w}}^{l,*}_k \tilde{\mathbf{x}}^1_k - \mathbf{w}^{u,*}_k \mathbf{x}^1_k) \\
&= \Delta \mathbf{b}^{l,*}_j + \textstyle\sum_{k=1}^{m} \left(\tilde{\mathbf{w}}^{l,*}_k (\mathbf{x}^1_k + \Delta^1_k) - \mathbf{w}^{u,*}_k \mathbf{x}^1_k\right) \\
&= \Delta \mathbf{b}^{l,*}_j + \textstyle\sum_{k=1}^{m} \left((\tilde{\mathbf{w}}^{l,*}_k - \mathbf{w}^{u,*}_k) \mathbf{x}^1_k + \tilde{\mathbf{w}}^{l,*}_k \Delta^1_k\right)
\end{aligned}$$

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 $\tilde{\mathbf{w}}^{l,*}_k = 2^{F^*} \hat{\mathbf{w}}^{l,*}_k$.

Given a quantized input $\hat{\mathbf{x}}$ of the QNN $\widehat{\mathcal{N}}$, the input difference of two networks is $2^{-F_{in}} \hat{\mathbf{x}} - \mathbf{x} = (2^{-F_{in}} \mathcal{C}^{\text{ub}}_h - 1)\mathbf{x}$. Therefore, we have $\Delta^1_k = \tilde{\mathbf{x}}^1_k - \mathbf{x}^1_k = 2^{-F_{in}} \hat{\mathbf{x}}^1_k - \mathbf{x}^1_k = (2^{-F_{in}} \mathcal{C}^{\text{ub}}_h - 1)\mathbf{x}$. Then, the lower bound of difference can be reformulated as follows using the input variables of the DNN:

$$\Delta l^{i,*}_{j,s} = \Delta \mathbf{b}^{l,*}_j + \textstyle\sum_{k=1}^{m} (-\mathbf{w}^{u,*}_k + 2^{-F_{in}} \mathcal{C}^{\text{ub}}_h \tilde{\mathbf{w}}^{l,*}_k) \mathbf{x}^1_k.$$

Similarly, we can reformulated the upper bound $\Delta u^{i,*}_{j,s}$ as follows using the input variables of the DNN:

$$\Delta u^{i,*}_{j,s} = \Delta \mathbf{b}^{u,*}_j + \textstyle\sum_{k=1}^{m} (-\mathbf{w}^{l,*}_k + 2^{-F_{in}} \mathcal{C}^{\text{ub}}_h \tilde{\mathbf{w}}^{u,*}_k) \mathbf{x}^1_k$$

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

V. Evaluation

We have implemented our method QEBVerif into an end-to-end tool written in Python, where we use Gurobi [35] as

TABLE I: Benchmarks for QNNs and DNNs.

ARCH	QNNs $Q=4$	$Q=6$	$Q=8$	$Q=10$	DNNs
P1: 1blk_100	96.38%	96.79%	96.77%	96.74%	96.92%
P2: 2blk_100	96.01%	97.04%	97.00%	97.02%	97.07%
P3: 2blk_512	96.69%	97.41%	97.35%	97.36%	97.36%
P4: 3blk_100	95.53%	96.66%	96.59%	96.68%	96.71%

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 2.80GHz CPU and 1 TB main memory. We allow Gurobi to use up to 24 threads. The time limit for each task is 1 hour.

We aim to answer the following research questions:

RQ1. How efficient and effective is the DRA in QEBVerif, compared with the baseline method?

RQ2. How efficient and effective is the MILP-based verification method as a complementary solution in QEBVerif?

RQ3. Whether the quantization error bound is correlated to the robustness of QNNs?

A. Benchmarks

We first train 4 DNNs with different architectures using the MNIST dataset [40] as image classifiers. Then, we build 16 QNNs from these DNNs, following a *post-training quantization scheme* [37] and using quantization configurations $\mathcal{C}_{in} = \langle +, 8, 8\rangle$, $\mathcal{C}_w = \langle \pm, Q, Q-1\rangle$, $\mathcal{C}_b = \langle \pm, Q, Q-2\rangle$, $\mathcal{C}_h = \langle +, Q, Q-2\rangle$ for each DNN, where $Q \in \{4, 6, 8, 10\}$. Details of all the networks are given in Table I. 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. For example, P2 names the architecture of $784 \times 100 \times 100 \times 10$. Columns 2-6 list the accuracy of these networks. Hereafter, we denote by Px-y the QNN using the architecture Px and quantization bit size $Q = y$, and by Px-Full the DNN of architecture Px. We can observe that all the 20 networks have more than 95% accuracy.

B. **RQ1**: *Effectiveness and Efficiency of the DRA*

To answer **RQ1**, we first implement the baseline using existing state-of-the-art reachability analysis methods for QNNs and DNNs. Specifically, the baseline uses the symbolic interval analysis of DeepPoly [36] to compute the output intervals for a DNN, and uses concrete interval analysis of [28] to compute the output intervals for a QNN. After that, it computes quantization error intervals via interval subtraction. Remark that no existing methods can directly verify quantization error bounds and the methods in [29]–[31] are not applicable in our setting. Finally, we compare the quantization error intervals computed by the baseline against ours, i.e., DRA, using DNNs Px-Full and QNNs Px-y for $x \in \{1, 2, 3, 4\}$ and $y \in \{4, 6, 8, 10\}$. We randomly select 30 input samples from the test set of the MNIST dataset that can be correctly classified by all the 20 networks in Table I. We set radius $r = 3$ for each input sample, resulting in total 30 different input regions, i.e., 30 tasks for each pair of DNN and QNN of same architecture.

Table II reports the results for all the non-input neurons. Column 2 lists different analysis methods, where "QEBVerif (Con)" is Algorithm 2, "QEBVerif (sym-)" uses the pure symbolic difference computation approach (cf. Section IV-C) and "QEBVerif (sym)" is the same as "QEBVerif (sym-)" except that Algorithm 3 is used to compute the output difference intervals of the hidden neurons instead of the abstract transformers for the ReLU and Clamp functions. Columns (H_Diff) and (OutDiff) give the average sum length of the difference intervals of all the non-input neurons of the predicted class for the 30 verification tasks, where the best ones (i.e., most accurate intervals) are highlighted in blue. Columns (Time) list the average computation time for the 30 tasks in seconds.

We can observe that "QEBVerif (sym)" produces the most accurate difference intervals of both hidden neurons and output neurons for almost all the tasks, except for P1-8 and P1-10 where "QEBVerif (Con)" performs better on the intervals for the output neurons. Unsurprisingly, 'QEBVerif (sym)" is less efficient than the others but is still in the same order of magnitude. We also find that "QEBVerif (Con)" performs even worse than the baseline method for larger QNNs, such as P2-4, P3-4 and P4-4 when the quantization bit size is small. It is because, (1) the rounding error of lower bit size is larger and (2) the concrete rounding error intervals added to hidden neurons are accumulated by this conservative method which becomes larger with the increase of the QNN size, compared to the baseline approach where we directly do the interval subtraction. By comparing "QEBVerif (sym)' with 'QEBVerif (sym-)", we observe that the better performance of "QEBVerif (sym)' comes from not only the abstract transformers but the activation transformation designed in Algorithm 3 which outperforms the abstract transformers for the ReLU and Clamp functions. Remark that the output layer does not use activation functions, so their difference intervals, i.e., the quantization error intervals, computed by "QEBVerif (sym)' and 'QEBVerif (sym-)" are exactly the same.

C. **RQ2**: *Effectiveness of Efficiency of Formal Verification*

To answer **RQ2**, we evaluate QEBVerif on QNNs Px-y for $x \in \{1, 2\}$, $y \in \{4, 6, 8, 10\}$, and P3-z for $z \in \{4, 8\}$, as well as DNNs correspondingly. We use the same 30 input regions as in Section V-B. We set the error bound $\epsilon = \{1, 2, 4, 6, 8\}$, resulting 150 verification tasks for each pair of DNN and QNN of same architecture.

Table III shows the verification results of QEBVerif within 1 hour per task. Columns (#Verified) and (Time) give the number of successfully verified tasks and average verification time of all the solved tasks in seconds, respectively. Columns (DRA) give the verification results using only the differential reachability analysis, i.e., QEBVerif (sym). Columns (DRA + MILP) 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. Columns (DRA + MILP + Diff) are similar to Columns (DRA + MILP) except that

TABLE II: Differential Reachability Analysis by QEBVerif vs. Baseline Method.

Q	Method	P1			P2			P3			P4		
		H_Diff	OutDiff	Time	H_Diff	OutDiff	Time	H_Diff	OutDiff	Time	H_Diff	OutDiff	Time
4	Baseline	64.45	7.02	0.78	220.9	20.27	1.54	470.1	22.69	12.17	551.6	47.75	2.38
	QEBVerif (Con)	32.86	6.94	0.80	249.8	33.44	1.55	1,399	98.56	12.10	614.5	56.86	2.41
	QEBVerif (sym)	32.69	3.14	1.33	134.9	7.11	2.91	365.3	11.11	24.20	313.8	14.90	5.07
	QEBVerif (sym-)	34.88	3.14	1.33	144.3	7.11	2.89	395.4	11.11	23.72	358.7	14.90	5.07
6	Baseline	68.94	7.89	0.78	249.5	24.25	1.53	612.2	31.67	11.89	616.2	54.66	2.39
	QEBVerif (Con)	10.33	2.19	0.79	102.8	14.74	1.54	516.2	34.62	11.97	492.9	57.50	2.41
	QEBVerif (sym)	10.18	1.46	1.35	55.73	3.11	2.86	158.5	3.99	22.86	131.3	5.33	4.73
	QEBVerif (sym-)	13.75	1.46	1.36	62.90	3.11	2.83	173.2	3.99	23.43	142.6	5.33	4.68
8	Baseline	69.15	7.95	0.78	251.6	24.58	1.52	620.6	32.43	11.90	623.1	55.42	2.38
	QEBVerif (Con)	4.27	0.89	0.79	42.6	6.54	1.54	206.9	13.85	11.99	341.7	43.26	2.41
	QEBVerif (sym)	4.13	1.02	1.35	34.01	2.14	2.83	96.26	2.39	22.89	82.9	3.48	4.61
	QEBVerif (sym-)	8.66	1.02	1.36	42.16	2.14	2.82	113.3	2.39	22.91	93.40	3.48	4.61
10	Baseline	69.18	7.96	0.78	252.0	24.63	1.52	620.4	32.40	12.23	624.0	55.55	2.40
	QEBVerif (Con)	2.72	0.56	0.79	26.32	4.28	1.54	103.6	7.03	12.34	267.3	35.08	2.42
	QEBVerif (sym)	2.61	0.92	1.36	28.59	1.91	2.82	81.08	2.01	23.36	71.33	3.96	4.58
	QEBVerif (sym-)	7.46	0.92	1.36	37.09	1.91	2.82	99.30	2.01	23.24	81.96	3.06	4.57

TABLE III: Verification Results of QEBVerif.

Arch	Q	DRA		DRA + MILP		DRA + MILP + Diff	
		#Verified	Time	#Verified	Time	#Verified	Time
P1	4	88	1.34	150	21.32	150	31.50
	6	130	1.37	148	2.56	149	27.51
	8	136	1.36	150	3.08	150	3.45
	10	139	1.37	150	42.88	150	125.7
P2	4	49	2.91	132	95.75	133	143.4
	6	88	2.87	138	46.13	138	16.49
	8	108	2.84	145	96.76	146	19.50
	10	112	2.86	145	134.6	145	9.19
P4	4	1	5.07	86	171.6	85	167.6
	8	86	4.58	122	137.7	135	184.3

linear constraints of the difference intervals of hidden neuron got from DRA are added into the MILP encoding.

We observe that, although DRA successfully verified most of tasks, our MILP-based verification method can help further verify many tasks on which DRA fails. 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., 15 tasks more in total. Note that, we use medium-sized QNNs and choose only two quantization bit sizes for P4-Full as evaluation benchmarks of this experiment for the sake of time and computing resource.

D. **RQ3**: *Correlation of Quantization Error and Robustness*

To answer **RQ3**, we first use QEBVerif to verify a set of properties $\Psi = \{P^c(\mathcal{N}, \widehat{\mathcal{N}}, \hat{\mathbf{x}}, r, \epsilon)\}$ such that:

- $\mathcal{N} =$ P1-Full, $\widehat{\mathcal{N}} \in$ {P1-4, P1-8};
- $\hat{\mathbf{x}} \in \mathcal{X}$ and $\mathcal{X}$ is the same as above;
- $r \in \{3, 5, 7\}$;
- $\epsilon \in \Omega = \{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0\}$.

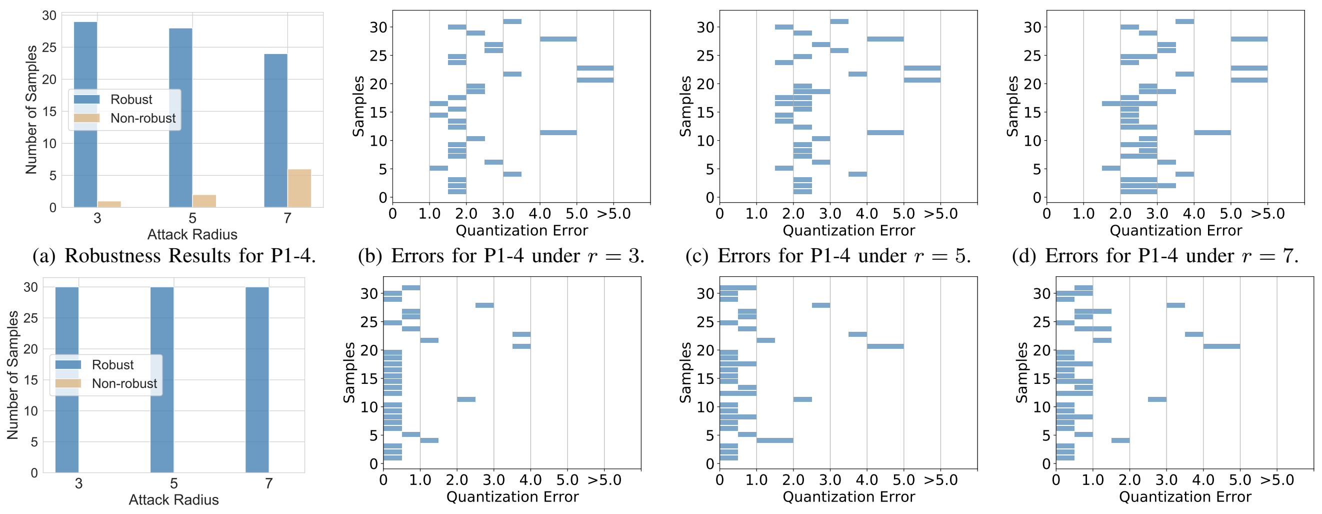
We solve all the above tasks and process all the verification results to obtain the tightest quantization error interval $[a, b]$ for each input region such that $a, b \in \Omega$. It allows us to obtain intervals that are tighter than those obtained via DRA. Finally, we implemented a robustness verifier for QNNs in a way similar to [39] to check the robustness of P1-4 and P1-8 w.r.t. the input regions given in Ψ.

Fig. 4 gives the experimental results. The blue (resp. yellow) bars in Figs. 4(a) and 4(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 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 more easily cause significantly 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 suggest that it is indeed possible to apply our method to compute the quantization error bound and use it as a guideline to identify the best quantization scheme which balances the size of the model and its robustness. We leave the task of identifying a practical threshold for the quantization error bound as a promising future work.

E. *Threats to Validity*

Our method is designed for verifying the quantization error between DNNs and their quantized counterparts which are typically deployed in safety-critical resource-constrained devices. Hence, the quantization bit sizes and the number of neurons are not very large. In such a context, the completeness of the verification method matters. In this work, we focus on the networks of a feed-forward architecture with the ReLU activation function. There are other architectures and activation functions. We leave them as future work.

The potential external threat is the selection of evaluation subjects. To mitigate this threat, we use 4 DNNs and 16 QNNs



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

of various architectures. Specifically, we train 4 DNNs on the MNIST dataset and build another 16 QNNs following a post-training quantization scheme. We remark that although there exist other ways to get QNNs, e.g., quantization-aware-training [37], how to choose a training method for a QNN is out of the scope of this work. We also randomly select 30 samples from the test set of the MNIST dataset for evaluation, and by setting varied values for error threshold, we can evaluate our tool on a huge set of verification tasks. For the effectiveness of QEBVerif on networks trained on other datasets (than MNIST), we leave it for future exploration.

VI. Related Work

While there is a large and growing body of work on quality assurance techniques for neural networks including testing (e.g., [15]–[23], [41]–[45]) and formal verification (e.g., [36], [46]–[66]), they typically target real-valued neural networks, i.e., DNNs. Though testing techniques are often effective in finding violations, they cannot prove their absence. Existing DNN formal verification methods are not efficient in verifying quantization error bound. In this section, we mainly discuss the existing verification techniques for quantized neural networks.

Early work on formal verification of QNNs typically focus on 1-bit quantized neural networks (i.e., BNNs) [24], [25], [27], [67]–[69]. Narodytska et al. [24] 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. [25] 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 [68], [69] via a BDD learning-based method [70]. Zhang et al. [27] 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. [8] 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. [28] explored several heuristics to improve the efficiency and scalability of [8]. Very recently, [39] proposed an MILP-based verification method for QNNs which outperforms the SMT-based methods. Though these works can directly verify QNNs or BNNs, they cannot verify quantization error bound.

There are also some work focusing on exploring properties of two neural networks which are most closely related to our work. Paulsen et al. [29], [30] proposed differential verification methods to verify two DNNs with the same network topology. This idea has been extended to handle recurrent neural networks [31]. The difference between [29]–[31] 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 program was used to analyze the different behaviours of DNNs and *fully* QNNs [14]. Different from our work focusing on verification, they aim at generating a upper bound for the worst-case error induced by quantization. Furthermore, [14] only scales to 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 1818 neurons.

VII. Conclusion

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 that QEBVerif is more efficient than the baseline method. 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, such as convolutional neural networks and recurrent neural networks.

References

[1] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. Li, “Large-scale video classification with convolutional neural networks,” in *Proceedings of 2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, 2012.

[3] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1532–1543.

[4] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding,” in *Proceedings of the 4th International Conference on Learning Representations*, 2016.

[5] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, “Fixed point quantization of deep convolutional networks,” in *Proceedings of the 33nd International Conference on Machine Learning*, 2016, pp. 2849–2858.

[6] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, and J. Yan, “Differentiable soft quantization: Bridging full-precision and low-bit neural networks,” *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4851–4860, 2019.

[7] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[8] M. Giacobbe, T. A. Henzinger, and M. Lechner, “How many bits does it take to quantize your neural network?” in *Proceedings of the 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2020, pp. 79–97.

[9] D. Zhang, J. Yang, D. Ye, and G. Hua, “Lq-nets: Learned quantization for highly accurate and compact deep neural networks,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 365–382.

[10] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, “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*, 2019, pp. 4350–4359.

[11] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort, “Up or down? adaptive rounding for post-training quantization,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 7197–7206.

[12] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, “Robust physical-world attacks on deep learning visual classification,” in *Proceedings of 2018 IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1625–1634.

[13] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, “Deep neural network compression for aircraft collision avoidance systems,” *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 3, pp. 598–608, 2019.

[14] J. Li, R. Drummond, and S. R. Duncan, “Robust error bounds for quantised and pruned neural networks,” in *Learning for Dynamics and Control*. PMLR, 2021, pp. 361–372.

[15] N. Carlini and D. A. Wagner, “Towards evaluating the robustness of neural networks,” in *Proceedings of the 2017 IEEE Symposium on Security and Privacy*, 2017, pp. 39–57.

[16] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, “Deepgauge: multi-granularity testing criteria for deep learning systems,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 120–131.

[17] S. Ma, Y. Liu, W. Lee, X. Zhang, and A. Grama, “MODE: automated neural network model debugging via state differential analysis and input selection,” in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 175–186.

[18] K. Pei, Y. Cao, J. Yang, and S. Jana, “Deepxplore: Automated whitebox testing of deep learning systems,” in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[19] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, “Concolic testing for deep neural networks,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 109–119.

[20] Y. Tian, K. Pei, S. Jana, and B. Ray, “Deeptest: automated testing of deep-neural-network-driven autonomous cars,” in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 303–314.

[21] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, “Deephunter: a coverage-guided fuzz testing framework for deep neural networks,” in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 146–157.

[22] A. Odena, C. Olsson, D. G. Andersen, and I. J. Goodfellow, “Tensorfuzz: Debugging neural networks with coverage-guided fuzzing,” in *Proceedings of the 36th International Conference on Machine Learning*, 2019, pp. 4901–4911.

[23] X. Xie, L. Ma, H. Wang, Y. Li, Y. Liu, and X. Li, “Diffchaser: Detecting disagreements for deep neural networks,” in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019, pp. 5772–5778.

[24] N. Narodytska, S. P. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh, “Verifying properties of binarized deep neural networks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, pp. 6615–6624.

[25] T. Baluta, S. Shen, S. Shinde, K. S. Meel, and P. Saxena, “Quantitative verification of neural networks and its security applications,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1249–1264.

[26] G. Amir, H. Wu, C. W. Barrett, and G. Katz, “An SMT-based approach for verifying binarized neural networks,” *CoRR*, vol. abs/2011.02948, 2020.

[27] Y. Zhang, Z. Zhao, G. Chen, F. Song, and T. Chen, “BDD4BNN: A bdd-based quantitative analysis framework for binarized neural networks,” in *Proceedings of the 33rd International Conference on Computer Aided Verification*, 2021, pp. 175–200.

[28] T. A. Henzinger, M. Lechner, and D. Zikelic, “Scalable verification of quantized neural networks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 3787–3795.

[29] B. Paulsen, J. Wang, and C. Wang, “Reludiff: Differential verification of deep neural networks,” in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 714–726.

[30] B. Paulsen, J. Wang, J. Wang, and C. Wang, “NeuroDiff: scalable differential verification of neural networks using fine-grained approximation,” in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 784–796.

[31] S. Mohammadinejad, B. Paulsen, J. V. Deshmukh, and C. Wang, “DiffRNN: Differential verification of recurrent neural networks,” in *Proceedings of the 19th International Conference on Formal Modeling and Analysis of Timed Systems*, 2021, pp. 117–134.

[32] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to interval analysis*. Siam, 2009, vol. 110.

[33] P. Cousot and R. Cousot, “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*, 1977, pp. 238–252.

[34] A. Lomuscio and L. Maganti, “An approach to reachability analysis for feed-forward ReLU neural networks,” *CoRR*, vol. abs/1706.07351, 2017.

[35] Gurobi, “A most powerful mathematical optimization solver,” https://www.gurobi.com/, 2018.

[36] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, “An abstract domain for certifying neural networks,” *Proceedings of the ACM on Programming Languages (POPL)*, vol. 3, pp. 41:1–41:30, 2019.

[37] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, “A white paper on neural network quantization,” *arXiv preprint arXiv:2106.08295*, 2021.

[38] QEBVerif, https://github.com/QEBVerif/Data, 2022.

[39] S. Mistry, I. Saha, and S. Biswas, “An MILP encoding for efficient verification of quantized deep neural networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (Early Access)*, 2022.

[40] Y. LeCun and C. Cortes, “Mnist handwritten digit database,” 2010.

[41] I. Shumailov, Y. Zhao, R. Mullins, and R. Anderson, "To compress or not to compress: Understanding the interactions between adversarial attacks and neural network compression," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 230–240, 2019.

[42] T. Baluta, Z. L. Chua, K. S. Meel, and P. Saxena, "Scalable quantitative verification for deep neural networks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering*. IEEE, 2021, pp. 312–323.

[43] S. Webb, T. Rainforth, Y. W. Teh, and M. P. Kumar, "A statistical approach to assessing neural network robustness," in *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[44] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Trans. Software Eng.*, vol. 48, no. 2, pp. 1–36, 2022.

[45] Z. Zhao, G. Chen, J. Wang, Y. Yang, F. Song, and J. Sun, "Attack as defense: characterizing adversarial examples using robustness," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 42–55.

[46] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *Proceedings of the 22nd International Conference on Computer Aided Verification*, 2010, pp. 243–257.

[47] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis*, 2017, pp. 269–286.

[48] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Proceedings of the 29th International Conference on Computer Aided Verification*, 2017, pp. 97–117.

[49] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *Proceedings of the 31st International Conference on Computer Aided Verification*, 2019, pp. 443–452.

[50] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *Proceedings of the 29th International Conference on Computer Aided Verification*, 2017, pp. 3–29.

[51] S. Gokulanathan, A. Feldsher, A. Malca, C. W. Barrett, and G. Katz, "Simplifying neural networks using formal verification," in *Proceedings of the 12th International Symposium NASA Formal Methods*, 2020, pp. 85–93.

[52] P. Ashok, V. Hashemi, J. Kretínský, and S. Mohr, "Deepabstract: Neural network abstraction for accelerating verification," in *Proceedings of the 18th International Symposium on Automated Technology for Verification and Analysis*, 2020, pp. 92–107.

[53] Y. Y. Elboher, J. Gottschlich, and G. Katz, "An abstraction-based framework for neural network verification," in *Proceedings of the 32nd International Conference on Computer Aided Verification*, 2020, pp. 43–65.

[54] P. Yang, R. Li, J. Li, C. Huang, J. Wang, J. Sun, B. Xue, and L. Zhang, "Improving neural network verification through spurious region guided refinement," in *Proceedings of 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, J. F. Groote and K. G. Larsen, Eds., 2021, pp. 389–408.

[55] W. Liu, F. Song, T. Zhang, and J. Wang, "Verifying relu neural networks from a model checking perspective," *J. Comput. Sci. Technol.*, vol. 35, no. 6, pp. 1365–1381, 2020.

[56] X. Guo, W. Wan, Z. Zhang, M. Zhang, F. Song, and X. Wen, "Eager falsification for accelerating robustness verification of deep neural networks," in *Proceedings of the 32nd IEEE International Symposium on Software Reliability Engineering*, 2021, pp. 345–356.

[57] J. Liu, Y. Xing, X. Shi, F. Song, Z. Xu, and Z. Ming, "Abstraction and refinement: Towards scalable and exact verification of neural networks," *CoRR*, vol. abs/2207.00759, 2022.

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

[59] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI^2: safety and robustness certification of neural networks with abstract interpretation," in *Proceedings of the 2018 IEEE Symposium on Security and Privacy*, 2018, pp. 3–18.

[60] J. Li, J. Liu, P. Yang, L. Chen, X. Huang, and L. Zhang, "Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification," in *Proceedings of the 26th International Symposium on Static Analysis*, 2019, pp. 296–319.

[61] R. Li, J. Li, C. Huang, P. Yang, X. Huang, L. Zhang, B. Xue, and H. Hermanns, "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*, 2020, pp. 1630–1634.

[62] G. Singh, R. Ganvir, M. Püschel, and M. T. Vechev, "Beyond the single neuron convex barrier for neural network certification," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2019, pp. 15 072–15 083.

[63] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. T. Vechev, "Fast and effective robustness certification," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2018, pp. 10 825–10 836.

[64] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *Proceedings of the 27th USENIX Security Symposium*, 2018, pp. 1599–1614.

[65] H. Tran, S. Bak, W. Xiang, and T. T. Johnson, "Verification of deep convolutional neural networks using imagestars," in *Proceedings of the International Conference on Computer Aided Verification*, 2020, pp. 18–42.

[66] H. Tran, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis of deep neural networks," in *Proceedings of the 3rd World Congress on Formal Methods*, 2019, pp. 670–686.

[67] A. Choi, W. Shi, A. Shih, and A. Darwiche, "Compiling neural networks into tractable boolean circuits," in *Proceedings of the AAAI Spring Symposium on Verification of Neural Networks*, 2019.

[68] A. Shih, A. Darwiche, and A. Choi, "Verifying binarized neural networks by local automaton learning," in *Proceedings of the AAAI Spring Symposium on Verification of Neural Networks*, 2019.

[69] ——, "Verifying binarized neural networks by angluin-style learning," in *Proceedings of the 2019 International Conference on Theory and Applications of Satisfiability Testing*, 2019, pp. 354–370.

[70] A. Nakamura, "An efficient query learning algorithm for ordered binary decision diagrams," *Inf. Comput.*, vol. 201, no. 2, pp. 178–198, 2005.