Quantization with Guaranteed Floating-Point Neural Network Classifications

ANAN KABAHA, Technion, Israel
DANA DRACHSLER COHEN, Technion, Israel

Despite the wide success of neural networks, their computational cost is very high. Quantization techniques reduce this cost, but it can result in changing the classifications of the original floating-point network, even if the training is quantization-aware. In this work, we rely on verification to design correction systems that detect classification inconsistencies at inference time and eliminate them. The key idea is to overapproximate the space of inconsistent inputs with their maximal classification confidence. The main challenge in computing this confidence is that it involves analyzing a quantized network, which introduces a very high degree of nonlinearity, over all possible inputs. We propose `CoMPAQt`, an algorithm for computing this confidence. `CoMPAQt` relies on a novel encoding of quantization in mixed-integer linear programming (MILP), along with customized linear relaxations to reduce the high complexity. To prune the search space, it ties the computations of the quantized network and its floating-point counterpart. Given this confidence, we propose two correction mechanisms. The first mechanism guarantees to return the classification of the floating-point network and relies on networks with increasing bit precisions. The second mechanism mitigates classification inconsistencies by an ensemble of quantized networks. We evaluate our approach on MNIST, ACAS-Xu, and tabular datasets over fully connected and convolutional networks. Results show that our first correction mechanism guarantees 100% consistency with the floating-point network's classifications while reducing its computational cost by 3.8x, on average. Our second mechanism reaches an almost perfect consistency guarantee in our experiments while reducing the computational cost by 4.1x. Our work is the first to provide a formal guarantee on the classification consistency of a quantized network.

CCS Concepts: • **Theory of computation → Program analysis**; **Program verification**; • **Software and its engineering → Formal methods**; • **Computing methodologies → Neural networks**.

Additional Key Words and Phrases: Neural Network Verification, Neural Network Quantization

ACM Reference Format:
Anan Kabaha and Dana Drachsler Cohen. 2025. Quantization with Guaranteed Floating-Point Neural Network Classifications. *Proc. ACM Program. Lang.* 9, OOPSLA2, Article 340 (October 2025), 28 pages. https://doi.org/10.1145/3763118

1 Introduction

Neural network classifiers have achieved remarkable success in various applications. However, their execution exhibits high computational complexity, leading to significant power consumption and large memory allocations, making them difficult to deploy on resource-constrained devices such as mobile phones and edge devices [Goodfellow et al. 2016; Li et al. 2019; Lin et al. 2020; Liu et al. 2021; Luo et al. 2023; Tan and Le 2019]. To address these challenges, many works leverage *quantization*, a technique that reduces the precision of a neural network's weights and activations from floating-point to lower-bit precisions (e.g., 8-bit or 12-bit) [Banner et al. 2019; Chauhan et al. 2023; Dong et al. 2019; Hubara et al. 2021; Jacob et al. 2018; Shen et al. 2024; Wang et al. 2018;

Authors' Contact Information: Anan Kabaha, Technion, Haifa, Israel, anan.kabaha@campus.technion.ac.il; Dana Drachsler Cohen, Technion, Haifa, Israel, ddana@ee.technion.ac.il.

ACM 2475-1421/2025/10-ART340
https://doi.org/10.1145/3763118

Yao et al. 2024; Yuan et al. 2023; Zhao et al. 2023; Zhuang et al. 2018]. Quantization significantly reduces power consumption, memory usage, and computation time. However, it also modifies the parameters of the floating-point network. This modification can cause the quantized network to change the classification of some inputs compared to the floating-point network, undermining its reliability. For example, an 8-bit quantized network for ACAS-Xu [Julian et al. 2018] can predict paths that are very far (over *13 miles* apart) from the paths of the floating-point network (Section 6).

The main quantization approaches are post-training quantization and quantization-aware training. Post-training quantization (PTQ) employs quantization after training a floating-point network [Chauhan et al. 2023; Shen et al. 2024]. Some of these approaches check several quantization schemes and return a quantized network with the best accuracy. Quantization-aware training (QAT) integrates quantization into the training process of a floating-point network, enabling the trained network to better adapt to the quantization performed after training completes [Jacob et al. 2018; Wang et al. 2018; Zhuang et al. 2018]. While these approaches aim to mitigate the impact of quantization on the network's classification, they have no formal guarantee. In particular, their effectiveness is typically evaluated empirically by comparing the quantized classifier's performance against its floating-point counterpart on a test set — and even on this limited set, their performance is not always perfect. In many cases, it is critical to understand and guarantee the consistency of quantized networks with their floating-point counterparts over a broader range of inputs, especially those not represented in the test set. This is particularly important in safety-critical applications for mobile and edge devices (e.g., health monitors, advanced driver assistance systems, and autonomous vehicles), where networks face inputs that may differ from those included in the test set.

To address this, several works study quantization reliability under input distribution shifts, noise, or input augmentations [Hu et al. 2022; Yuan et al. 2023]. Others propose verifiers that analyze the local robustness of a quantized network in given neighborhoods of inputs [Huang et al. 2024; Lin et al. 2021; Zhang et al. 2022]. Yet these works offer limited guarantees (whether empirical or formal) over a constrained set of inputs. Beyond analyzing the reliability of quantized networks, it is crucial to *correct* classifications that are inconsistent because of the quantization. In this work, we address the question: *Given a network classifier and a quantization scheme, can we identify all classification inconsistencies of the quantized network and mitigate them?*

We propose a new property over neural network classifiers that overapproximates *all* classification inconsistencies caused by the quantization with their maximal classification confidence. We call this property the **Q***uantized* **E***rror compared to* **F***loating-point* (QEF) bound. It provides an effective way to identify whether *any* (unseen) input may be classified differently due to the quantization. Beyond the formal guarantee for every possible input, this property can be leveraged by different correction mechanisms to reduce and even eliminate the quantization classification inconsistencies. This opens a new paradigm for quantizing networks via formal verification analysis.

Computing the QEF bound is highly challenging because it involves reasoning about a quantized network, whose computations are highly nonlinear. Generally, the quantized computations transform the output of every neuron into a piecewise linear function whose number of linear pieces grows exponentially with the bit precision. As a reference point, a (floating-point) neuron performing the popular ReLU activation function computes a piecewise linear function over two linear pieces, and this function type alone makes the analysis of safety properties over a neural network NP-hard [Katz et al. 2017]. In contrast, an 8-bit neuron performing the popular ReLU activation function computes a piecewise linear function consisting of 2^8 linear pieces. Additionally, QEF is a global property, requiring to consider every possible input (not restricted to a dataset).

We introduce CoMPAQt, an algorithm for computing the QEF bound. CoMPAQt relies on several ideas. First, it relies on a novel mixed-integer linear programming (MILP) encoding for quantization. For each quantized ReLU activation function, our encoding introduces two boolean variables to precisely encode the bounded range of the quantized ReLU and overapproximates the discrete values in the quantized ReLU's range with linear constraints bounding a parallelogram. To reduce this encoding's complexity, `CoMPAQt` also employs a trapezoid linear relaxation if its overapproximation error is not too high. `CoMPAQt` further prunes the search space by tying the computations of the quantized network and its floating-point counterpart. This is obtained by deriving linear constraints over respective neurons by bound propagation and MILP optimization.

Based on the `QEF` bounds, `CoMPAQt` constructs corrected access to the quantized network. We propose two correction mechanisms. The first correction mechanism employs adaptive quantization precision refinement, relying on networks with increasing precision levels. At inference, the corrected access looks for the lowest-bit quantized network whose classification is consistent with the floating-point network. This mechanism guarantees to return consistent classifications, for *every* possible input, while minimizing the computational cost. To the best of our knowledge, this is the first quantization mechanism which is guaranteed to be fully consistent with the floating-point network for all inputs. This correction mechanism is especially suitable for settings where deploying high bit precision networks is feasible, but reducing the computational cost (e.g., power consumption) is critical. The second correction mechanism relies on an ensemble of low bit precision networks. At inference, it looks for a quantized network whose classification is consistent. If none is found, it returns a classification and signals the user that it may be inconsistent. In practice, this mechanism significantly improves the classification consistency. Its advantage is that it relies solely on lower bit precision networks, thus it is better suited for extremely resource-constrained settings.

We evaluate `CoMPAQt` on two tabular datasets, the MNIST image dataset, and the ACAS-Xu system. We evaluate on fully connected and convolutional networks. We compare to PTQ and QAT. Our results show that the adaptive precision refinement mechanism obtains 100% consistency guarantee for every input while reducing the computational cost of the floating-point network by 3.8x. The baselines do not obtain perfect consistency on the test set but have lower cost than `CoMPAQt` by 1.4x. The low-bit ensemble mechanism achieves 99.994% consistency on the test set, reduces the computational cost by 4.1x, and its overhead compared to the baselines is 1.03x. We show similar results for out-of-distribution inputs generated by common image corruptions (e.g., snow). Lastly, we evaluate `CoMPAQt` on a case study of ACAS-Xu, showing that it precisely predicts the floating-point network's flight trajectories, unlike the baselines that deviate by 65.83 miles.

2 Background on Quantized Networks

In this section, we provide background on quantized networks.

Quantization is a technique for reducing the computational time and memory consumption of neural networks [Banner et al. 2019; Jacob et al. 2018; Shen et al. 2024; Zhuang et al. 2018]. Its idea is to approximate continuous floating-point values of weights and activations with discrete values. This is achieved by lower bit precisions such as 8-bit or 16-bit (denoted INT8 or INT16). This process significantly reduces the network's memory consumption and power consumption.

Quantization is typically defined over three parameters: the scale factor s, the zero-point p, and the bit-width b [Nagel et al. 2021]. The scale factor and zero-point parameters define how to map a floating-point value to a discrete integer. The range of integers, called the *quantization grid*, depends on the bit-width parameter b. An example of a grid is the unsigned integer set $\{0, \dots, 2^b - 1\}$. Given (s, p, b), a **quantization step** maps a real number x to an integer: $x_q = \mathtt{clamp}(\lfloor x/s \rceil + p)$, where $\lfloor \cdot \rceil$ rounds to the nearest integer and `clamp` ensures that the quantized value is within the quantization grid. For example, for the grid $\{0, \dots, 2^b - 1\}$, $\mathtt{clamp}_{0,2^b-1}(y)$ returns 0 if $y < 0$, it returns $2^b - 1$, if $y > 2^b - 1$; otherwise, it returns y. A **dequantization step** maps a quantized value x_q to a real number: $\hat{x} = s(x_q - p)$. For example, given a real-valued input $x = 3.7$, a scale factor $s = 0.1$, a zero-point $p = 10$, a bit-width $b = 8$, and the grid $\{0, \dots, 255\}$, the quantized value is computed as: $x_q = \texttt{clamp}_{0,255}(\lfloor 3.7/0.1 \rceil + 10) = 47$. The dequantized value is: $\hat{x} = 0.1 \cdot (47 - 10) = 3.7$, accurately recovering the original value in this case. Quantization can introduce two types of errors: rounding errors and clipping errors. A rounding error stems from the composition of $s(\texttt{clamp}_{0,2^b-1}(\lfloor x/s \rceil + p) - p)$ and it is within the range $[-s/2, s/2]$. For example, for $x = 3.75$, $x_q = \texttt{clamp}_{0,255}(\lfloor 3.75/0.1 \rceil + 10) = 48$. The dequantized value is $\hat{x} = 0.1 \cdot (48 - 10) = 3.8$, introducing a quantization error of 0.05. A clipping error occurs when the quantized value is clamped. For example, for $x = 30$, we have $x_q = \texttt{clamp}_{0,255}(\lfloor 30/0.1 \rceil + 10) = 255$ and $\hat{x} = 0.1 \cdot (255 - 10) = 24.5$, introducing a quantization error of 5.5.

Various quantization approaches have been proposed for neural networks, including ones that quantize the weights, the activation outputs, or both. We focus on the popular approach that quantizes the weights and activation outputs [Nagel et al. 2021]. To formally define it, we first define a neural network. A neural network is a directed layered graph, whose edges are associated with weights and its nodes (*neurons*) are associated with parameters called biases. Figure 1(a) shows an example of a network. The set of edges and the computations associated with them depend on the layer's type. We focus on fully connected layers, but our implementation also supports convolutional layers and is extensible to max-pooling [LeCun et al. 1998] and residual layers [He et al. 2016]. In a fully connected layer, a neuron has an incoming edge from every neuron in the previous layer. We denote by $z_{m,k}$ the $k \in [k_m]$ neuron in layer $m \in [L]$, and denote by $z_{0,k}$ for $k \in [d]$ a neuron in the input layer. The neuron's weights are denoted by $w_{m,k,k'}$, for all $k' \in [k_{m-1}]$, and its bias is denoted by $b_{m,k}$. Its computation consists of an affine computation and a nonlinear computation, defined by an activation function. We focus on the ReLU activation. Formally, a neuron computes the affine function: $\hat{z}_{m,k} = b_{m,k} + \sum_{k' \in [k_{m-1}]} w_{m,k,k'} \cdot z_{m-1,k'}$, and then invokes ReLU: $z_{m,k} = ReLU(\hat{z}_{m,k}) = \max(0, \hat{z}_{m,k})$. We note that ReLU is not invoked at the last layer.

In a floating-point network, all weights, biases, and activation outputs are real numbers. Figure 1(a) shows an example of a floating-point network. The computation of the network is executed by propagating a given input layer-by-layer. At every layer, (1) the inputs are multiplied by their weights (denoted on the edges) and summed $z_\Sigma = \sum_{k'} w_{k'} \cdot z_{k'}$, then (2) the bias is added $\hat{z} = b_z + z_\Sigma$ (denoted by +), and (3) ReLU is invoked $z = ReLU(\hat{z})$ (denoted by the ReLU sign). To illustrate the computation, consider the input $(0.8, 0)$. First, it is multiplied by the weights (0.9 and 0.4 for the first neuron) and summed, then the bias is added (-0.63 for the first neuron) and ReLU is invoked.

In a quantized network, the weights, the input layer, and the activation outputs are quantized, but the bias and the affine computations are not quantized [Nagel et al. 2021]. We focus on symmetric uniform quantization with zero-points $p = 0$, however our algorithms extend to other quantization schemes. A quantization scheme for a neural network consists of a scale factor s_m for the inputs to layer m, a scale factor $s_{m,W}$ for the weights of layer m, and a bit-width b. That is, a quantization scheme is a pair (S, b) where S is a set of pairs $(s_m, s_{m,W})$ for every layer m. We assume the inputs' grid is $\{0, \dots, 2^b - 1\}$ and the weights' grid is $\{-2^{b-1}, \dots, 2^{b-1} - 1\}$. Figure 1(b) shows an example of a quantized network for the network in Figure 1(a), where, for simplicity, $\forall m.\ s_m = s_{m,W} = 1/15$ and $b = 4$. The computation of the quantized network is executed layer-by-layer. At every layer, (1) the inputs are quantized (denoted by the steps icon), (2) multiplied by their weights and summed (their sum is in floating-point), (3) the sum is dequantized (denoted by the straight line), then (4) the bias is added, and (5) ReLU is invoked. To illustrate, consider the input $(0.8, 0)$. First, it is quantized to $(12, 0)$, then multiplied by the weights (14 and 6 for the first neuron) and summed (i.e., $12 \cdot 14 + 0 \cdot 6$). The weighted sum is dequantized and added the bias (-0.63 for the first neuron). Then, ReLU is invoked and its output passes to the next layer, which performs the same process. We observe that in the setting of analyzing quantized networks, the dequantization and

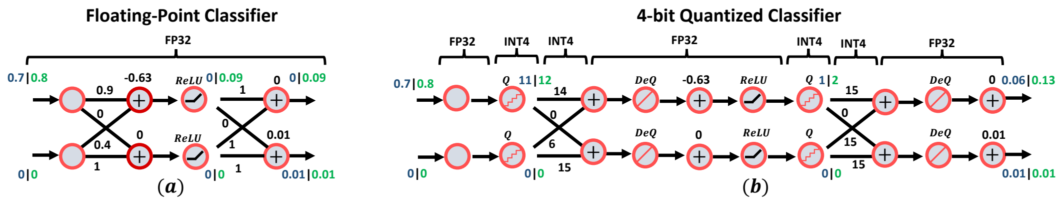


Fig. 1. (a) A floating-point network N and (b) its quantized network N_q. N_q is classification inconsistent with N for the input $(0.7, 0)$ (its computations for both networks are shown on the left, in blue). N_q is classification consistent with N for $(0.8, 0)$ (its computations are shown on the right, in green).

the computation of the weighted sum are commutative. Accordingly, we define the computations of a quantized network as follows. For every $k \in [d]$, we define $\zeta_{0,k}$ as the dequantization of the quantized input: $\zeta_{0,k} = s_0 \cdot \texttt{clamp}_{0,2^b-1}(\lfloor x_k/s_0 \rceil)$. For every neuron k in layer $m \in [L]$, we denote by $\hat{\zeta}_{m,k}$ the dequantization of the weights composed with the weighted sum and bias addition, i.e., $\hat{\zeta}_{m,k} = b_{m,k} + s_{m,W} \cdot \sum_{k'} w^q_{m,k,k'} \cdot \zeta_{m-1,k'}$, where $w^q_{m,k,k'} = \texttt{clamp}_{0,2^b-1}(\lfloor w_{m,k,k'}/s_{m,W} \rceil)$, and we denote by $\zeta_{m,k}$ the dequantization of the quantized ReLU: $\zeta_{m,k} = s_m \cdot \texttt{clamp}_{0,2^b-1}(\lfloor ReLU(\hat{\zeta}_{m,k})/s_m \rceil)$.

In the following, we denote a floating-point network by N and its quantized counterpart by N_q. We note that there are different ways to compute a quantized network, e.g., by post-training quantization (PTQ) [Banner et al. 2019; Hubara et al. 2021; Zhang et al. 2023], quantization-aware training (QAT) [Jacob et al. 2018; Zhuang et al. 2018], or mixed-precision strategies [Chauhan et al. 2023; Dong et al. 2019; Yang et al. 2024]. The exact process of network's quantization is orthogonal to our work, we merely require to have a floating-point network and a quantization scheme.

3 Our Property: Quantization Inconsistent Classifications

In this section, we present a property for quantized network classifiers, which captures the inputs that a quantized network may classify differently from its floating-point network, due to quantization errors. We begin with defining classification inconsistency, then motivate and define our property, and finally discuss the challenge of proving it.

Classification inconsistencies. We focus on neural networks that act as classifiers. A classifier N maps an input $x \in [0,1]^d$ to a score vector over a set of classes C. The classification of N for x is the class maximizing the score: $c = \arg\max(N(x))$. Because the role of a classifier is to assign a class, the exact values in the score vector are less important, which enables to tolerate some quantization rounding and clipping errors. We say a quantized classifier is classification consistent for an input if it classifies the input the same as the floating-point network. Formally, given a quantized network N_q, its floating-point network N, and an input x, we say that N_q is *classification consistent* with N for x if $\arg\max(N_q(x)) = \arg\max(N(x))$. For example, for N_q and N in Figure 1, N_q is classification consistent with N for $x = (0.8, 0)$, since they classify x the same. However, N_q is classification *inconsistent* with N for $x = (0.7, 0)$, since they classify it differently.

Goal. Our goal is to identify *all* classification inconsistencies at inference time (i.e., given an input for N_q) and correct them. There are two impractical ways to identify all classification inconsistencies. First, a static exhaustive search, where a preprocessing step propagates every input through both networks and stores the inconsistent inputs. However, this is infeasible since the space of inputs is too large in practice. Second, a dynamic approach, where at inference time, given an input to classify, the input is propagated through both the quantized network and the floating-point network. However, this approach defeats the purpose of using a quantized network to save resources.

Key idea. We identify a condition on the output of a quantized network $\varphi(N_q(x))$ that, if satisfied, implies that N_q may be classification inconsistent with N for x. The advantage of such a condition is that it can be efficiently checked at inference time, with negligible overhead since either way x is propagated through N_q. If the condition holds, we invoke a correction mechanism, which may be less resource efficient. This enables us to keep the efficiency of the quantized network for inputs that N_q is *guaranteed* to be classification consistent. Conceptually, this condition partitions the input space into two parts: one containing all inputs that introduce classification inconsistencies (but may contain other inputs) and another part that does not contain any input that introduces a classification inconsistency. Our condition builds on the observation that the differences of a quantized network and its floating-point network stem from their differences in the decision boundaries. That is, the classification inconsistent inputs are close to the decision boundary of the quantized network. This is illustrated in Figure 2(a). In this example, we consider a synthetic dataset over inputs $(x_1, x_2) \in [0,1]^2$ classified to a class in $C = \{0,1\}$. We train a floating-point classifier N and compute a 4-bit quantized network N_q. The figure shows the decision boundaries of N (in black line) and of N_q (in red line). The classification inconsistencies are relatively close to the decision boundaries. In particular, the maximal distance of any input classified differently by N and N_q from the decision boundary of N_q is relatively small. We capture it by the *classification confidence* of the quantized network, which is an arithmetic expression over its output vector. We then define a condition $\varphi(N_q(x))$ that overapproximates the classification inconsistencies with their maximal classification confidence, called the QEF bound. We next provide the formal definitions.

The QEF *bound.* Given a quantized network N_q, an input x and a class $c \in C$, the classification confidence is the difference between the score N_q assigns to c given x and the maximal score of the other classes: $\mathcal{C}^c_{N_q}(x) = N_q(x)_c - \max_{c' \neq c} N_q(x)_{c'}$. Given a quantized network N_q, its floating-point network N and a class $c \in C$, the *Quantized Error compared to Floating-Point (*QEF*) bound $d_c \in \mathbb{R}^+$* is the maximal classification confidence of any input that N_q classifies as c and N classifies differently:

$$d_c = \max \mathcal{C}^c_{N_q}(x) \text{ subject to } \arg\max(N_q(x)) = c \wedge \arg\max(N(x)) \neq c \quad (1)$$

Given d_c, our condition to identify inputs which may introduce classification inconsistency for c is:

$$\varphi_c(N_q(x)) = \mathcal{C}^c_{N_q}(x) \leq d_c$$

If $d_c = 0$, this condition is not satisfied by any input that N_q classifies as c. Namely, every input that N_q classifies as c is also classified as c by N and no correction is needed. If $d_c = \max\{\mathcal{C}^c_{N_q}(x) \mid x \in [0,1]^d\}$ (the maximum confidence of N_q for c), then every input that N_q classifies as c may introduce a classification inconsistency and all these inputs are required to be corrected. Note that our bound is sound but imprecise: it overapproximates the set of inputs introducing classification inconsistency. We call the input space satisfying the condition $\varphi_c(N_q(x))$ the *non-*QEF space, since these inputs may introduce classification inconsistency, and the input space not satisfying $\varphi_c(N_q(x))$ the QEF space, since these inputs are classification consistent.

Illustration. Figures 2(a)–(d) illustrate the inputs satisfying our condition, for different values of bounds d'_c defining the condition $\mathcal{C}^c_{N_q}(x) \leq d'_c$ (i.e., not just the maximal bounds d_c that are the QEF bounds). The background of these figures is split into colored and pink. The pink background shows the non-QEF space with respect to d'_c, while the colored background shows the QEF space. The colored background is a heatmap of the inputs' classification confidences. The darker the color, the lower the classification confidence of N_q. In Figure 2(a), $d'_c = 0$, thus all inputs are in the QEF space. However, this is an incorrect (unsound) partitioning of the input domain, since this QEF space includes inputs that are classified differently by N_q and N (i.e., they are classification inconsistent).

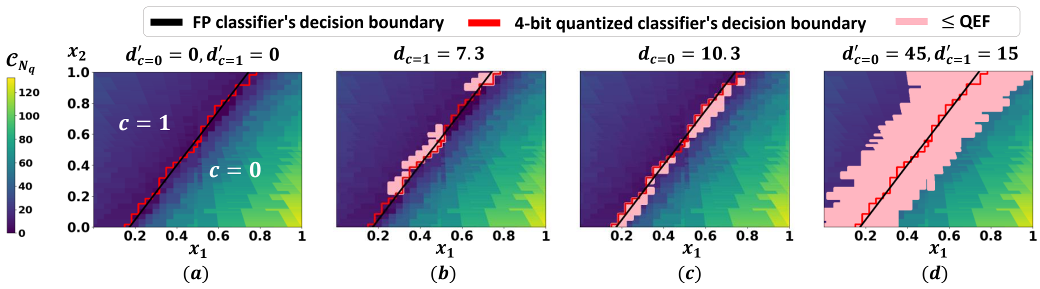


Fig. 2. The inputs satisfying $\mathcal{C}^c_{N_q}(x) \le d'_c$ (pink background) for different values of $d'_c \in \{0, d_{c=1}, d_{c=0}, 15, 45\}$, where $d_{c=1}, d_{c=0}$ are the QEF bounds of classes 0 and 1. Values lower than d_c underapproximate the classification inconsistencies of class c, while values greater or equal to d_c overapproximate them.

In Figure 2(b), $d'_{c=1} = d_{c=1}$ (the QEF bound of class c_1). This bound is the maximal bound at which there is an input classified as c_1 by N_q but as c_0 by N. All inputs in its QEF space are classified as c_1 by both networks. In Figure 2(c), $d'_{c=0} = d_{c=0}$ (the QEF bound of class c_0). Note that these QEF bounds are tight: there exists an input x classified as c by N_q with confidence equal to d_c but as the other class by N *and* there is no x' that is classified as c with confidence higher than d_c that is classified as the other class by N. Still, the non-QEF space overapproximates the classification inconsistent inputs: Figure 2(b) shows inputs classified by N_q and N as $c = 1$ whose confidence is lower than $d_{c=1}$. Larger bounds than the QEF bounds provide a sound partitioning of the input space, but they increase the overapproximation error, leading to unnecessary corrections. For example, in Figure 2(d), the non-QEF space soundly overapproximates the classification inconsistent inputs of both classes, but its overapproximation error is very high: nearly half of the input space is in the non-QEF input space, even though most of these inputs are classified the same by N_q and N.

Challenges. Computing the QEF bound of every class is challenging because it is a *global* property over all $x \in [0, 1]^d$, and over two networks with nonlinear computations. This problem's complexity is exponential in the bit-width of the quantized network multiplied by its number of neurons.

4 Overview on Computing the QEF Bound

In this section, we overview the computation of the QEF bound. We phrase it as mixed-integer linear programming (MILP) with customized linear relaxations to balance precision and scalability. To further scale, we add constraints over the quantized network and its floating-point network.

QEF *with MILP.* The QEF bound (Equation (1)) is a constrained optimization over two networks. Many verifiers for neural networks rely on MILP for certifying local robustness of floating-point networks [Müller et al. 2022; Singh et al. 2019b; Tjeng et al. 2019] and quantized networks [Huang et al. 2024; Lin et al. 2021], global robustness [Kabaha and Drachsler-Cohen 2024; Wang et al. 2022a,b], and privacy properties [Kabaha and Drachsler-Cohen 2025; Reshef et al. 2024]. However, our property poses a new challenge: it is a global property and over quantization ($x_q = \texttt{clamp}_{0,2^b-1}(\lfloor x/s \rceil)$). This requires encoding in MILP new operations: the steps operation $\lfloor\cdot\rceil$ and the clipping operation $\texttt{clamp}_{0,2^b-1}$. We note that the dequantization step ($\hat{x} = s \cdot x_q$) is directly expressible as a linear constraint. It is possible to precisely encode the steps operation and the clipping operation in MILP. However, the complexity is too high to be effective in practice because precisely encoding $\lfloor\cdot\rceil$ requires b boolean variables (to capture its 2^b steps) and precisely encoding $\texttt{clamp}_{0,2^b-1}$ requires two boolean variables (to capture its three cases: values below 0, above $2^b - 1$, or in between).

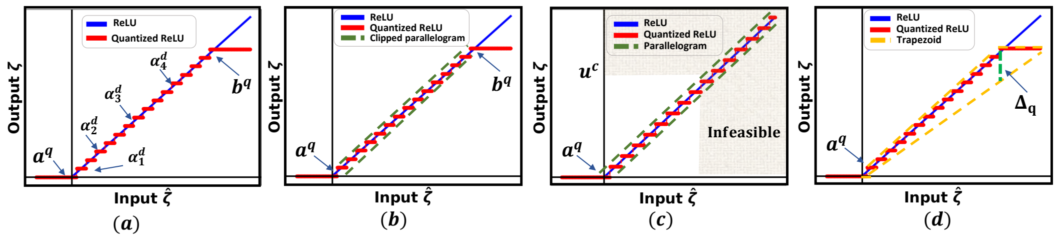


Fig. 3. Our linear relaxations for quantization. (a) A precise MILP encoding introduces $b + 2$ boolean variables for each dequantization of a quantized ReLU $\zeta_{m,k}$. (b) Our parallelogram linear relaxation bounds the steps operation instead of the b booleans. (c) If the maximal possible value is at most $s \cdot (2^b - 1)$, we can eliminate the boolean b^q. (d) Otherwise, we can overapproximate the active part with a trapezoid.

Figure 3(a) illustrates the dequantization of the quantized ReLU $\zeta_{m,k}$ as a function of its input $\hat{\zeta}_{m,k}$ and the $b + 2$ booleans required to encode all its piecewise linear parts. In contrast, encoding a floating-point ReLU computation requires *one* boolean variable. This means that our problem's complexity is $O(2^{(b+2)\cdot|N|})$ where $|N|$ is the number of neurons in the quantized network.

Steps linear relaxation. We next introduce a customized *linear relaxation* for the steps operation, instead of the MILP encoding that relies on b boolean variables. Linear relaxations bound nonlinear computations using lower and upper linear constraints. They have been shown to boost the analysis of local robustness verifiers in floating-point networks [Müller et al. 2022; Singh et al. 2019a,b; Wang et al. 2021]. Commonly, linear relaxations are linear constraints over the input to the ReLU neuron and its real-valued lower and upper bounds. An example of a linear relaxation to $z = ReLU(\hat{z})$, where $\hat{z} \in [l, u]$ for $l, u \in \mathbb{R}$, is the triangle $z \geq 0$, $z \geq \hat{z}$ and $z \leq \frac{u(\hat{z}-l)}{u-l}$ [Ehlers 2017]. Existing linear relaxations are too coarse-grained for a quantized ReLU, resulting in a very high overapproximation error and thus a very loose QEF bound, triggering unnecessary corrections. Instead, we perform a linear relaxation to the steps operation with a parallelogram bounding the rounding errors (that are in $[-s/2, s/2]$). Additionally, we precisely encode the clipping operation with two boolean variables a^q and b^q. Figure 3(b) shows this parallelogram (dashed green lines) and the two boolean variables.

Infeasible clipping. Prior MILP verifiers reduce their complexity by eliminating booleans, which is possible if some linear pieces are infeasible. We extend this approach to our setting and identify infeasible outputs of ReLU. Like prior approaches, our identification relies on real-valued lower and upper bounds for the weighted sum $\hat{\zeta}$ of every neuron (which are computed as part of the analysis either way). Given bounds $[l, u]$ of $\hat{\zeta}$, if $u \leq s \cdot (2^b - 1)$, then clipping to $s \cdot (2^b - 1)$ is never executed, enabling us to eliminate the boolean variable b^q. We note that l and u are also used to define the parallelogram. Figure 3(c) illustrates the case when clipping to $s \cdot (2^b - 1)$ is infeasible.

Trapezoid linear relaxation. If clipping to $s \cdot (2^b - 1)$ is possible, we propose another way to eliminate the boolean variable b^q, using a trapezoid linear relaxation for overapproximating the steps operation and the step of $s \cdot (2^b - 1)$. Figure 3(d) illustrates our trapezoid linear relaxation. This linear relaxation is used only if its overapproximation error (denoted by Δ_q) is smaller than a predefined threshold. If the error is too high, we precisely encode the clipping operation as described before. We define the error Δ_q as the maximal difference between the value of a quantized step of $\hat{\zeta}$ and its farthest value in the trapezoid (shown in green line in Figure 3(d)).

Tying the floating-point and quantized networks. The QEF bound is defined over a quantized network and its floating-point network. While they have different computations, the computations

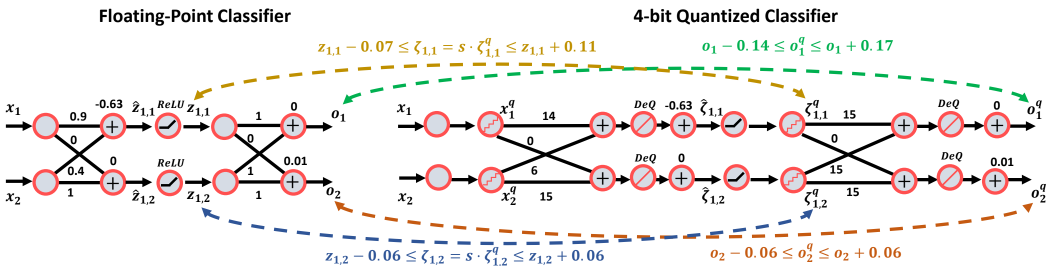

Floating-Point Classifier
4-bit Quantized Classifier
$z_{1,1} - 0.07 \le \zeta_{1,1} = s \cdot \zeta^q_{1,1} \le z_{1,1} + 0.11$
$o_1 - 0.14 \le o^q_1 \le o_1 + 0.17$
$z_{1,2} - 0.06 \le \zeta_{1,2} = s \cdot \zeta^q_{1,2} \le z_{1,2} + 0.06$
$o_2 - 0.06 \le o^q_2 \le o_2 + 0.06$
ReLU
DeQ

Fig. 4. Differences of neurons in a floating-point network N (left) and a quantized network N_q (right).

are related. Thus, to prune the search space, we compute this relation and add it as linear constraints. Technically, we bound the differences of respective neurons of the two networks within a real-valued interval and add it as a constraint. Bounding the differences of respective neurons has been proposed before, for analyzing the network's computation over two slightly different inputs [Banerjee et al. 2024; Wang et al. 2022a,b] and for analyzing two similar networks' computation over the same inputs [Kabaha and Drachsler-Cohen 2025]. However, our setting requires a new definition of these differences because of the new quantization operations and the lower bit precision of the weights and the layers' inputs. In particular, the difference introduced at each input neuron due to quantization is bounded within the interval $[-s/2, s/2]$, the difference due to the clipping operation is bounded within $[\min(0, s \cdot (2^b - 1) - u), 0]$, and the difference between each quantized weight and its floating-point counterpart is a real-valued constant. We compute the differences of respective neurons with two approaches: bound propagation, which is fast but imprecise, and two MILPs, which are precise but slower. Bound propagation propagates these differences through the entire network to bound the maximal difference of respective neurons. Our bound propagation is sound and can only prune the search space and not add overapproximation error or reduce the number of boolean variables. Our two MILPs are limited by a relatively short timeout. The differences of respective neurons are obtained by intersecting the bounds of the two approaches. This provides a natural precision-scalability tradeoff: for some neurons, MILP is fast enough to provide tighter bounds, while in other cases, MILP is slower and thus the bound propagation provides tighter bounds. We formally define this computation in Section 5.1.4.

Figure 4 illustrates the bound propagation for a 4-bit quantized network N_q (with $s = 1/15$) and its floating-point network N. Given an input x to N and its quantized version x^q to N_q, the difference between x and x^q is $\Delta^s = [-s/2, s/2] = [-0.033, 0.033]$. The difference of a quantized weight and its counterpart is a constant, for example the distance of the weight $14/15$ from 0.9 is $\Delta^w_{1,1,1} = 0.033$ and the distance of the weight $6/15$ from 0.4 is $\Delta^w_{1,1,2} = 0$. Bound propagation passes the input differences through the weights of the first layer. For example, the weighted sum of the top neuron in N's first layer is: $\hat{z}_{1,1} = 0.9x_1 + 0.4x_2 - 0.63$. We express the weighted sum of the respective neuron in N_q in terms of $\hat{z}_{1,1}$ plus their difference $\hat{\zeta}_{1,1} = (0.9 + \Delta^w_{1,1,1})(x_1 + \Delta^s) + (0.4 + \Delta^w_{1,1,2})(x_2 + \Delta^s) - 0.63 = \hat{z}_{1,1} + \Delta^w_{1,1,1}(x_1 + \Delta^s) + 0.9\Delta^s + 0.4\Delta^s$. We obtain a bounding interval on $\hat{\zeta}_{1,1} - \hat{z}_{1,1}$ by substituting the lower and upper bounds of $\Delta^w_{1,1,1}$, x_1, and Δ^s in this expression. We pass this interval in ReLU, in the quantization (i.e., $\zeta^q_{1,1} = \texttt{clamp}_{0,2^b-1}(\lfloor ReLU(\hat{\zeta}_{1,1})/s \rceil)$), and in the dequantization (i.e., $\zeta_{1,1} = s \cdot \zeta^q_{1,1}$) and obtain $\zeta_{1,1} - z_{1,1} \in [-0.07, 0.11]$ (the computations are defined in Section 5.1.4). Next, we compute these differences by solving two MILPs, one minimizing $\zeta_{1,1} - z_{1,1}$ and the other maximizing it. They return $\zeta_{1,1} - z_{1,1} \in [-0.07, 0.11]$. We intersect these intervals and add the constraints $\zeta_{1,1} \ge z_{1,1} - 0.07$ and $\zeta_{1,1} \le z_{1,1} + 0.11$. Similarly, we proceed to the following layers.

5 CoMPAQt: A System for Guaranteed Classification Consistent Quantization

In this section, we describe CoMPAQt, our system that provides formally guaranteed classification consistent quantization. CoMPAQt first computes the QEF bound of a quantized network and then constructs a correction mechanism to guarantee classification consistency. We begin with the algorithm for computing the QEF bound and then present the correction mechanisms.

5.1 Algorithm for Computing the QEF Bound

In this section, we describe CoMPAQt's algorithm for computing the QEF bound, called CoMPAQt-QEF (Algorithm 1). It takes as input a floating-point network N, a quantization scheme (S, b), and a class c. It returns the QEF bound, which can be leveraged to mitigate classification inconsistencies, as described in Section 5.2. At high-level, it encodes the constrained optimization defining the QEF bound as MILP and submits this MILP to a MILP solver. CoMPAQt-QEF begins by computing the quantized network N_q (Line 1). Then, it constructs the MILP constraints. It begins with generating variables $x_1, \dots, x_d$ for capturing any input in the range $[0, 1]^d$ and storing the range constraints in I (Line 2). Then, it adds to P a MILP encoding of N, the constraints on the input layer $z_{0,k} = x_k$ for all k, and the constraints in I (Line 3). The encoding of N is the MILP introduced by MIPVerify [Tjeng et al. 2019], described in Section 5.1.1. Then, it initializes the constraints of the quantized network in $\mathcal{P}$ with I and the constraints on the input layer $\zeta_{0,k} = x_k + [-s_0/2, s_0/2]$ for all k (recall that $\zeta_{0,k}$ represents the dequantized version of $x_{k,0}$'s quantization) (Line 4). Then, it iterates over the layers and their neurons and encodes their computations. For every neuron, it first encodes the neuron's affine transformation into a variable $\hat{\zeta}_{m,k}$ (Line 7). Then, it computes $\hat{\zeta}_{m,k}$'s lower and upper bounds $l^q_{m,k}$ and $u^q_{m,k}$ (Line 8), similarly to MIPVerify, as described in Section 5.1.1. Then, it encodes the computation of $\zeta_{m,k}$ which is the dequantization of the quantized ReLU. This encoding has three possibilities (described in Figure 3). To pick among them, it first computes the maximal dequantized value u^c_m (Line 9), which depends on s_m and b of the quantization scheme. If $u^q_{m,k}$ is at most u^c_m, then no value is above u^c_m and so there is no clipping to the value $s_m \cdot (2^b - 1)$. In this case, CoMPAQt-QEF overapproximates $\zeta_{m,k}$ with the parallelogram relaxation and one boolean variable, as shown in Figure 3(c) and defined in Equation (2) (Line 10–Line 11). Otherwise, it computes the overapproximation error of the trapezoid bounding the steps function and the clipping to the maximum value (Line 13). If the error is too large, it overapproximates with a parallelogram and two boolean variables, as shown in Figure 3(b) and defined in Equation (3) (Line 14–Line 15). Otherwise, it overapproximates with a trapezoid and one boolean variable, as shown in Figure 3(d) and defined in Equation (4) (Line 16–Line 17). After encoding the neuron's function, CoMPAQt-QEF computes a relation over the neuron and its respective neuron in N (Line 18). After completing encoding N_q, CoMPAQt-QEF collects the constraints for N, N_q, and the QEF bound (described in Section 5.1.3), and submits to a MILP solver (Line 19). Finally, it returns the QEF bound (Line 20).

5.1.1 MIPVerify. We next describe the MILP encoding of MIPVerify [Tjeng et al. 2019], used in Line 3 and that we adapt for the quantized network. MIPVerify determines whether a neural network classifier N is locally robust in a neighborhood around an input x classified as c. It determines robustness if N classifies all inputs in the neighborhood as c. The neighborhood is encoded by intervals bounding the input entries (which CoMPAQt-QEF replaces with constraints allowing any input in Line 2). The input neurons are associated with real-valued variables $z_{0,k}$ for $k \in [d]$ and they are assigned the input entries: $z_{0,k} = x_k$. For every layer $m \in [L]$ and neuron $k \in [k_m]$, MIPVerify has two variables: (1) $\hat{z}_{m,k}$ for the pre-activation value storing the weighted sum and (2) $z_{m,k}$ for the activation output. The pre-activation $\hat{z}_{m,k}$ is encoded by a linear equality constraint capturing the weighted sum and the bias. To define the constraints of $z_{m,k}$, MIPVerify

```
Algorithm 1: CoMPAQt-QEF($N$, ($S = \{(s_m, s_{m,W}) \mid m \in [L]\}, b$), $c$)
  Input: A floating-point network $N$, a quantization scheme $(S, b)$, and a class $c$.
  Output: The QEF bound $d_c$.
1  $N_q \leftarrow$ quantize($N, (S, b)$)                               // The quantized classifier
2  $I \leftarrow \{x_k \leq 1, x_k \geq 0 \mid k \in [d]\}$                       // The (any) input variables
3  $P \leftarrow I \cup \{z_{0,k} = x_k \mid k \in [d]\} \cup$ MIPVerify_Build($N$)           // MILP for $N$
4  $\mathcal{P} \leftarrow I \cup \{\zeta_{0,k} \leq x_k + s_0/2, \zeta_{0,k} \geq x_k - s_0/2 \mid k \in [d]\}$   // MILP for $N_q$'s input layer
5  foreach $m \in [L]$ do                                        // Iterate over $N_q$'s layers
6      foreach $k \in [m_k]$ do                  // Iterate over the neurons in the layer
7          $\mathcal{P} = \mathcal{P} \cup \{\hat{\zeta}_{m,k} = b_{m,k} + s_{m,W} \cdot \sum_{k'=1}^{k_{m-1}} w^q{}_{m,k,k'} \cdot \zeta_{m-1,k'}\}$
8          $u^q_{m,k}, l^q_{m,k} \leftarrow$ compute_bounds($\mathcal{P}, \hat{\zeta}_{m,k}$)              // Compute neuron's bounds
9          $u^c_m \leftarrow s_m \cdot (2^b - 1)$                         // The maximal dequantized value
10         if $u^q_{m,k} \leq u^c_m$ then
11             $\mathcal{P} \leftarrow \mathcal{P} \cup$ ParallelogramRelaxation($\mathcal{P}, (S, b), m, k, l^q_{m,k}, u^q_{m,k}$)
12         else
13             $\Delta_q \leftarrow u^c_m - \frac{u^c_m}{u^q_{m,k} - s_m/2} \cdot (u^c_m - s_m/2)$             // Overapproximation error
14             if $\Delta_q \geq T_{trapezoid}$ then                                 // Error is too large
15                 $\mathcal{P} \leftarrow \mathcal{P} \cup$ ClippedParallelogram($\mathcal{P}, (S, b), m, k, l^q_{m,k}, u^q_{m,k}$)
16             else
17                 $\mathcal{P} \leftarrow \mathcal{P} \cup$ TrapezoidRelaxation($\mathcal{P}, (S, b), m, k, l^q_{m,k}, u^q_{m,k}$)
18         $\mathcal{P} \leftarrow \mathcal{P} \cup$ compute_relation($N, (S, b), N_q, P, \mathcal{P}, m, k$)
19 $d_c \leftarrow$ MILP_solver($P \cup \mathcal{P} \cup$ encode_QEF($P, \mathcal{P}$))
20 return $d_c$
```

computes lower and upper bounds, $l_{m,k}, u_{m,k} \in \mathbb{R}$, for $\hat{z}_{m,k}$. This is done by interval arithmetic or two MILPs. The MILPs consist of all constraints up to layer $m - 1$, the equality constraint of $\hat{z}_{m,k}$ and the objectives $l_{m,k} = \min \hat{z}_{m,k}$ and $u_{m,k} = \max \hat{z}_{m,k}$ (one objective for each MILP). After computing these bounds, MIPVerify determines the constraints for $z_{m,k}$ as follows. If $l_{m,k} \geq 0$, i.e., $z_{m,k}$ is always active in this neighborhood, it adds $z_{m,k} = \hat{z}_{m,k}$. If $u_{m,k} \leq 0$, i.e., $z_{m,k}$ is inactive, it adds $z_{m,k} = 0$. Otherwise, it introduces a boolean variable $a_{m,k} \in \{0, 1\}$ and four constraints, such that if a satisfying assignment for these constraints assigns $a_{m,k} = 1$, then $z_{m,k}$ is in its active state and otherwise $z_{m,k}$ is in its inactive state. The four constraints are:

$$z_{m,k} \geq 0, \qquad z_{m,k} \leq u_{m,k} \cdot a_{m,k}, \qquad z_{m,k} \geq \hat{z}_{m,k}, \qquad z_{m,k} \leq \hat{z}_{m,k} - l_{m,k}(1 - a_{m,k})$$

The first two constraints enforce the output 0 for the inactive case and the other two constraints enforce the output $\hat{z}_{m,k}$ for the active case. To check whether N classifies all inputs in the neighborhood as c, MIPVerify adds a constraint that the score of c is not the maximal score: $z_{L,c} - \max_{c' \neq c} z_{L,c'} \leq 0$, where max is encoded by the Big-M method [Vanderbei 1996], defining a boolean variable for each $c' \neq c$. N is locally robust at the neighborhood if and only if this MILP is infeasible.

5.1.2 Our MILP for quantized networks. In this section, we introduce our MILP encoding for a quantized network, defined by a network N and a scheme $(\{(s_m, s_{m,W}) \mid m \in [L]\}, b)$. Encoding the quantized network's parameters is straightforward: the biases $b_{m,k}$ of every $m \in [L]$, $k \in [k_m]$ are identical to the floating-point network's biases and the quantized weights $w^q_{m,k,k'}$ of every $m \in [L]$, $k \in [k_m]$, and $k' \in [k_{m-1}]$ are in lower precision compared to $w_{m,k,k'}$ but are still constants. Our encoding has similar real variables as N for the pre-activation part $\hat{\zeta}_{m,k}$ and for the activation part $\zeta_{m,k}$ (corresponding to $\hat{z}_{m,k}$ and $z_{m,k}$). The pre-activation output is encoded similarly to N except that it includes a dequantization of the weights: $\hat{\zeta}_{m,k} = b_{m,k} + s_{m,W} \cdot \sum_{k'=1}^{k_{m-1}} w^q{}_{m,k,k'} \cdot \zeta_{m-1,k'}$. The challenge is encoding $\zeta_{m,k} = s_m \cdot \texttt{clamp}_{0,2^b-1}(\lfloor ReLU(\hat{\zeta}_{m,k})/s_m \rceil)$. As discussed in Section 4 and illustrated in Figure 3, there are three possible encodings:

1) $u^q_{m,k} \le u^c_m$: If $\hat{\zeta}_{m,k}$ cannot exceed the maximal dequantized value, CoMPAQt-QEF adds a boolean variable $a^q_{m,k}$ for capturing the active and inactive cases, two constraints for enforcing 0 in the inactive case (when $a^q_{m,k} = 0$), and constraints for overapproximating the active case (when $a^q_{m,k} = 1$) by its bounding parallelogram:

$$\zeta_{m,k} \ge 0, \quad \zeta_{m,k} \le u^q_{m,k} \cdot a^q_{m,k} \tag{2a}$$

$$\zeta_{m,k} \ge l^q_{m,k}, \quad \zeta_{m,k} \ge \hat{\zeta}_{m,k} - s_m/2, \quad \zeta_{m,k} \le \hat{\zeta}_{m,k} + s_m/2 - l^q_{m,k} \cdot (1 - a^q_{m,k}) \tag{2b}$$

Lemma 5.1. *If $u^q_{m,k} \le u^c_m$, Equation (2) overapproximates the dequantization of the quantized ReLU.*

Proof. We split to cases by the boolean variable's assignment:

- $a^q_{m,k} = 0$: By Equation (2a), $\zeta_{m,k} = 0$. The other constraints hold.
- $a^q_{m,k} = 1$: Since $u^q_{m,k} \le u^c_m$, there is no clipping to u^c_m. Thus, $\zeta_{m,k}$ is $\hat{\zeta}_{m,k}$ plus a rounding error $[-s_m/2, s_m/2]$. This computation is bounded by the parallelogram: $\zeta_{m,k} \ge \max(0, l^q_{m,k})$, $\zeta_{m,k} \le u^q_{m,k}$, $\zeta_{m,k} \ge \hat{\zeta}_{m,k} - s_m/2$, and $\zeta_{m,k} \le \hat{\zeta}_{m,k} + s_m/2$, which hold when $a^q_{m,k} = 1$.

□

2) $u^q_{m,k} > u^c_m$: If the quantized ReLU may be clipped to $u^c_m = s_m \cdot (2^b - 1)$, CoMPAQt-QEF adds another boolean variable $b^q_{m,k}$. If $b^q_{m,k} = 0$, the ReLU output is determined by $a^q_{m,k}$: it is 0 if $a^q_{m,k} = 0$ and is bounded by the parallelogram if $a^q_{m,k} = 1$. If $b^q_{m,k} = 1$, the output is u^c_m. Our encoding is:

$$a^q_{m,k} \ge b^q_{m,k}, \quad \zeta_{m,n} \ge 0, \quad \zeta_{m,k} \le u^q_{m,k} \cdot a^q_{m,k}, \tag{3a}$$

$$\zeta_{m,k} \ge \hat{\zeta}_{m,k} - s_m/2 - (u^q_{m,k} - u^c_m) \cdot b^q_{m,k}, \qquad \zeta_{m,k} \le \hat{\zeta}_{m,k} + s_m/2 - l^q_{m,k} \cdot (1 - a^q_{m,k}), \tag{3b}$$

$$\zeta_{m,k} \ge l^q_{m,k}, \qquad \zeta_{m,k} \ge u^c_m \cdot b^q_{m,k}, \quad \zeta_{m,k} \le u^c_m \tag{3c}$$

Lemma 5.2. *The above constraints overapproximate the dequantization of the quantized ReLU.*

Proof. We split to cases by the boolean variables' assignments:

- $a^q_{m,k} = 0$: By Equation (3a), $b^q_{m,k} = 0$ and $\zeta_{m,k} = 0$. The other constraints hold.
- $a^q_{m,k} = 1$ and $b^q_{m,k} = 0$: $\zeta_{m,k}$ is bounded by the parallelogram whose correctness is proven in Lemma 5.1. The other constraints hold.
- $a^q_{m,k} = 1$ and $b^q_{m,k} = 1$: By Equation (3c), $\zeta_{m,k} = u^c_m$. The other constraints hold.

□

3) Trapezoid relaxation. To reduce the added complexity of the boolean $b^q_{m,k}$, we propose to bound the active state within a trapezoid, if the maximum difference between the clipped dequantized value and its corresponding overapproximated value is below a predefined threshold. This encoding is:

$$\zeta_{m,k} \geq 0, \qquad \zeta_{m,k} \leq u^c_m \cdot a^q_{m,k}, \tag{4a}$$

$$\zeta_{m,k} \geq \frac{u^c_m}{u^q_{m,k} - s_m/2} \cdot (\hat{\zeta}_{m,k} - s_m/2), \qquad \zeta_{m,k} \leq \hat{\zeta}_{m,k} + s_m/2 - l^q_{m,k}(1 - a^q_{m,k}) \tag{4b}$$

Lemma 5.3. *The above constraints overapproximate the dequantization of the quantized ReLU.*

Proof. We split to cases by the boolean variable's assignment:

- $a^q_{m,k} = 0$: By Equation (4a), $\zeta_{m,k} = 0$. The other constraints hold.
- $a^q_{m,k} = 1$: By $\zeta_{m,k} \leq u^c_m$, $\zeta_{m,k}$ is at most the maximal dequantized value. By $\zeta_{m,k} \geq 0$, $\zeta_{m,k}$ is at least the minimal dequantized value. By $\zeta_{m,k} \leq \hat{\zeta}_{m,k} + s_m/2$, $\zeta_{m,k}$ is below the steps function plus the maximal rounding error. By $\zeta_{m,k} \geq \frac{u^c_m}{u^q_{m,k} - s_m/2} \cdot (\hat{\zeta}_{m,k} - s_m/2)$, $\zeta_{m,k}$ is above the steps function minus the maximal rounding error.

□

5.1.3 `QEF` *bound as MILP.* We next present our MILP encoding for the `QEF` bound. This bound is defined over any input x within the range $[0,1]^d$. The input layer of N is defined as $z_{0,k} = x_k$, for all $k \in [d]$. The dequantization of the quantized input layer of N_q is bounded by: $\zeta_{0,k} \leq x_k + s_0/2$ and $\zeta_{0,k} \geq x_k - s_0/2$, for all $k \in [d]$. The network N is encoded as described in Section 5.1.1 and the quantized network N_q is encoded as described in Section 5.1.2, based on the choices described in Lines 10–17. The `QEF` bound's conditions are encoded as follows. The term $\mathcal{C}^c_N(x) \leq 0$ is encoded by $z_{L,c} - \max_{c' \neq c} z_{L,c'} \leq 0$, where max is expressed by the Big-M method [Vanderbei 1996], which introduces a large constant M and boolean variables $a_{c'}$ for every $c' \neq c$. For the term $\mathcal{C}^c_{N_q}(x) \geq d_c$, we introduce an optimization variable d_c and add constraints ensuring that N_q classifies x as the given class c with confidence of at least d_c, namely: $\zeta_{L,c} - \zeta_{L,c'} \geq d_c, \quad \forall c' \neq c$. Additionally, `CoMPAQt-QEF` adds relations ϕ_{rels} between respective neurons (Section 5.1.4). The objective function is the maximization of the `QEF` bound d_c. Overall, the complete MILP encoding is:

$$\max d_c \qquad \text{subject to} \tag{5a}$$

$$\phi_{rels}; \quad x \in [0,1]^d; \qquad \forall k.\ z_{0,k} = x_k; \qquad \forall k.\ \zeta_{0,k} \leq x_k + s_0/2; \qquad \forall k.\ \zeta_{0,k} \geq x_k - s_0/2 \tag{5b}$$

$$\forall c' \neq c.\ \zeta_{L,c} - \zeta_{L,c'} \geq d_c; \qquad \forall c' \neq c.\ z_{L,c} - z_{L,c'} \leq M \cdot (1 - \alpha_{c'}); \quad \sum_{c' \neq c} \alpha_{c'} \geq 1 \tag{5c}$$

$$\forall m > 0, \forall k. \qquad \hat{z}_{m,k} = b_{m,k} + \sum_{k'=1}^{k_{m-1}} w_{m,k,k'} \cdot z_{m-1,k'} \tag{5d}$$

$$\forall m > 0, \forall k. \quad z_{m,k} \geq 0; \quad z_{m,k} \geq \hat{z}_{m,k}; \quad z_{m,k} \leq u_{m,k} \cdot a_{m,k}; \quad z_{m,k} \leq \hat{z}_{m,k} - l_{m,k}(1 - a_{m,k}) \tag{5e}$$

$$\forall m > 0, \forall k. \quad \hat{\zeta}_{m,k} = b_{m,k} + s_{m,W} \sum_{k'=1}^{k_{m-1}} {w^q}_{m,k,k'} \cdot \zeta_{m-1,k'} \qquad + \text{constraints for } \zeta_{m,k} \text{ (Section 5.1.2)} \tag{5f}$$

5.1.4 Relational constraints. Lastly, we describe `compute_relation`, which returns relational constraints over the outputs of respective neurons (Line 18). It computes an interval bounding the neurons' difference $\zeta_{m,k} - z_{m,k} \in [\underline{\Delta}_{m,k}, \overline{\Delta}_{m,k}]$ and then creates the constraints $\zeta_{m,k} \geq z_{m,k} + \underline{\Delta}_{m,k}$ and $\zeta_{m,k} \leq z_{m,k} + \overline{\Delta}_{m,k}$. To derive these relations, it runs two computations: bound propagation, which is efficient but imprecise, and MILP optimization, which is slower but more precise. The running time of the MILP optimization is limited by a timeout T_{bound}. We denote the interval that the bound propagation returns by $[\underline{\Delta^P}_{m,k}, \overline{\Delta^P}_{m,k}]$ and the interval that the MILP optimization returns by $[\underline{\Delta^M}_{m,k}, \overline{\Delta^M}_{m,k}]$. Accordingly, `compute_relation` sets $\underline{\Delta}_{m,k} = \max(\underline{\Delta^M}_{m,k}, \underline{\Delta^P}_{m,k})$ and $\overline{\Delta}_{m,k} = \min(\overline{\Delta^M}_{m,k}, \overline{\Delta^P}_{m,k})$. This approach allows it to balance precision and efficiency.

Bound propagation. Bound propagation bounds the difference of $\zeta_{m,k} - z_{m,k}$ with interval arithmetic. The differences stem from the quantization of the inputs to $\zeta_{m,k}$ and their weights. For every $\zeta_{m,k}$, the difference with its floating-point ReLU value is bounded by the rounding error and the clipping to the maximal dequantized value u^c_m: $Q^z_{m,k} = \left[-\max\left(\frac{s_m}{2}, u^q_{m,k} - u^c_m\right), \frac{s_m}{2}\right]$. The difference of respective weights is $Q_{w_{m,k,k'}} = s_{m,W} \cdot w^q_{m,k,k'} - w_{m,k,k'}$. We define the difference interval $I^P_{m,k} = [\underline{\Delta^P}_{m,k}, \overline{\Delta^P}_{m,k}]$ inductively on the layer m. For $m = 0$ and $k \in [d]$, $I^P_{0,k} = [0, 0]$ and $Q^z_{0,k} = \left[-\frac{s_0}{2}, \frac{s_0}{2}\right]$ (since there is no clipping at the input layer). For every $m > 0, k \in [m_k]$:

$$I^P_{\hat{z}_{m,k}} = b_{m,k} + \sum_{k'=1}^{k_{m-1}} s_{m,W} \cdot w^q_{m,k,k'} \cdot \zeta_{m-1,k'} - \hat{z}_{m,k} =$$

$$b_{m,k} + \sum_{k'=1}^{k_{m-1}} (Q_{w_{m,k,k'}} + w_{m,k,k'}) \cdot (Q^z_{m-1,k} + I^P_{\hat{z}_{m-1,k'}} + z_{m-1,k'}) - \hat{z}_{m,k} =$$

$$\sum_{k'=1}^{k_{m-1}} (Q_{w_{m,k,k'}} + w_{m,k,k'}) \cdot (Q^z_{m-1,k} + I^P_{\hat{z}_{m-1,k'}}) + Q_{w_{m,k,k'}} \cdot z_{m-1,k'}$$

In the above equation, intervals consisting of one value $[z, z]$ are abbreviated to z. This expression is bounded by the real-valued lower and upper bounds of $z_{m-1,k'}$. For the interval of $I^P_{z_{m,k}}$, bounding the differences of ReLUs, we follow the definition of Kabaha and Drachsler-Cohen [2025]: $I^P_{z_{m,k}} = [-\max(0, -\underline{I_{\hat{z}_{m,k}}}), \max(0, \overline{I_{\hat{z}_{m,k}}})]$ (the quantization error is not added, since it is included in $I^P_{\hat{z}_{m,k}}$).

MILP optimization. Our second approach for computing the bounds relies on MILP optimization. It bounds $\zeta_{m,k} - z_{m,k}$ in the interval defined by: $\underline{\Delta^M}_{m,k} = \min_{P_{[m]}, \mathcal{P}_{[m]}}(\zeta_{m,k} - z_{m,k})$ and $\overline{\Delta^M}_{m,k} = \max_{P_{[m]}, \mathcal{P}_{[m]}}(\zeta_{m,k} - z_{m,k})$. Both MILPs consist of all constraints in P and $\mathcal{P}$ up to layer m. They are submitted to a MILP solver with a timeout T_{bound}. `CoMPAQt-QEF` relies on an anytime solver, which returns sound bounds: tight or suboptimal (if it reaches the timeout).

5.2 Correction Mechanisms to Mitigate Classification Inconsistencies

In this section, we describe two correction mechanisms to mitigate classification inconsistencies using our `QEF` bounds. The first mechanism, called *adaptive precision refinement* (`APR`), *guarantees* consistency with the floating-point network but may require higher bit precision. The second mechanism, called `QEF`-*Ensemble*, relies only on low-bit precision networks. It may not eliminate all classification inconsistencies, but it notifies the user when this occurs.

Adaptive precision refinement. The adaptive precision refinement (`APR`) correction mechanism takes as input a floating-point network N, quantization scale factors $S = \{(s_m, s_{m,W}) \mid m \in [L]\}$,

Algorithm 2: The Adaptive Precision Refinement Correction Mechanism

```
1  Function APR-Init($N, S, b_1, b_2, \dots, b_K$)
       Input: A floating-point network $N$, quantization scale factors $S$ and $K$ precision levels.
2      foreach $i \in [K]$ do
3          $N_q^{b_i} \leftarrow$ quantize($N, (S, b_i)$)
4          foreach $c \in C$ do
5              $d[i][c] \leftarrow$ CoMPAQt-QEF($N, (S, b_i), c$)

6  Function APR-Inference($x$)
       Input: An input $x$.
       Output: A classification consistent with $N$ minimizing the computational cost.
7      $i \leftarrow 1$
8      while $i \leq K$ do
9          $z_L \leftarrow N_q^{b_i}(x)$                      // Pass through the quantized network
10         $c \leftarrow \arg\max z_L$                                     // Predicted class
11         if $z_{L,c} - \max_{c' \neq c} z_{L,c'} > d[i][c]$ then return $c$            // QEF bound is met
12         foreach $j \in \{i+1, \dots, K\}$ do
13             if $z_{L,c} - \max_{c' \neq c} z_{L,c'} > d[j][c]$ then $i \leftarrow j$; break // QEF bound may be met
14             else if $j == K$ then $i \leftarrow K + 1$   // No QEF bound is expected to be met
15     return $\arg\max N(x)$            // Fallback to the floating-point network
```

and an increasing series of K precision levels $b_1 < b_2 < \cdots < b_K$ (e.g., 8-bit, 16-bit, 32-bit). It computes the QEF bounds of every class and every quantization scheme. At inference, upon any input x, it passes x through the lowest precision network $N_q^{b_1}$. If the classification confidence is above the corresponding QEF bound, it returns the predicted class. Otherwise, it refines the precision to the minimal b_i whose corresponding QEF bound is below the current classification confidence. Then, it passes x through $N_q^{b_i}$. If the classification confidence is above the corresponding QEF bound, it returns the predicted class. Otherwise, it refines the precision and continues similarly. If all confidences are below their QEF bounds, APR runs x through N and returns the predicted class. This operation guarantees that the returned class is *exactly* the class that N assigns to x even if x is not passed through N. Algorithm 2 shows the algorithm of APR. Its initialization takes as input a floating-point network, a set of quantization scale factors, and K precision levels. For each precision level, it constructs the quantized network, computes the QEF bound of every class, and stores them (Lines 1-5). Its inference takes an input x to the network. APR then iterates the quantized networks, in increasing precision level (Line 8). For each quantized network, it passes x through the network (Line 9) and identifies its predicted class c (Line 10). Then, it checks if the classification confidence of c is greater than its corresponding QEF bound (Line 11). If yes, it is *guaranteed* that N classifies x as c. Thus, c is returned. Otherwise, APR heuristically selects the next quantized network as the one whose QEF bound for c is lower than the current classification confidence (Lines 12–14). If none of the quantized networks is guaranteed to be classification consistent with N for x, APR falls back to the floating-point network, passes x through N, and returns its predicted class (Line 15). For example, consider three precision levels: INT8, INT12, and INT16, whose QEF bounds for c are: $d[\texttt{INT8}][c] = 10$, $d[\texttt{INT12}][c] = 5$, and $d[\texttt{INT16}][c] = 1$. Given an input x_0, APR passes it in the INT8 network which classifies it as c. Assume its classification confidence is $z_{L,c} - \max_{c' \neq c} z_{L,c'} = 2$, which is lower than the QEF bound making it insufficient to guarantee consistency for the INT8

Algorithm 3: Ensemble for Consistent Quantization

```
1  Function QEF-Ensemble-Init($N$, $\mathcal{T}$, $\mathcal{N}$, $\mathcal{D}$, $(S, b)$, $K$)
       Input: A floating-point network $N$, a training algorithm $\mathcal{T}$, a network architecture $\mathcal{N}$, a
              dataset $\mathcal{D}$, a quantization scheme $(S, b)$ and the ensemble size $K$.
2      foreach $i \in [K]$ do
3          if $i = 1$ then $N^1 \leftarrow N$
4          else $N^i \leftarrow \mathcal{T}(\mathcal{N}, \mathcal{D}, \text{random\_seed})$
5          $N^i_b \leftarrow$ quantize($N^i$, $(S, b)$)
6          foreach $c \in C$ do
7              $d[i][c] \leftarrow$ CoMPAQt-QEF($N^i$, $(S, b)$, $c$)

8  Function QEF-Ensemble-Inference($x$)
       Input: An input $x$.
       Output: A predicted class and a flag indicating if it is classification consistent.
9      foreach $i \in [K]$ do
10         $z_L \leftarrow N^i_b(x)$                                  // Pass through the quantized network
11         $c \leftarrow \arg\max z_L$                                               // Predicted class
12         if $z_{L,c} - \max_{c' \neq c} z_{L,c'} > d[i][c]$ then return ($c$, True)      // QEF bound is met
13     return ($\arg\max N^1_b(x)$, False)                          // Fallback to the original network
```

network. Since the INT12 network's bound is 5, APR does not pass x_0 through it. Since the INT16 network's bound is 1, APR passes x_0 through it. If the classification confidence is over 1, APR returns c, guaranteeing consistency with the floating-point network (i.e., the classification is unaffected by quantization). Otherwise, APR passes x_0 through the floating-point network and returns its predicted class, thereby APR trivially ensures a consistent classification.

By the operation of APR and the correctness of CoMPAQt-QEF, we get the following theorem:

Theorem 5.4. *For every $x \in [0, 1]^d$, APR returns the class c that N assigns to x while aiming to lower the computational cost.*

Ensemble of quantized networks. The ensemble of quantized network correction mechanism (QEF-Ensemble) leverages a network ensemble to mitigate classification inconsistencies while using only low-bit quantized networks. A network ensemble consists of a set of networks trained for the same task. Network ensembles are popular in many tasks, including adversarial robustness, where the ensemble aims at improving resilience against adversarial attacks [Pang et al. 2019; Strauss et al. 2017]. Its idea is to leverage the training variations to reduce the likelihood that an adversarial perturbation misleads all networks in the ensemble. Similarly, our idea is to rely on an ensemble to reduce the likelihood that an input is below the QEF bound of all networks in the ensemble. That is, the ensemble increases the robustness to classification inconsistency, with respect to some network in the ensemble. Our ensemble of quantized networks is generated by training floating-point networks with the original training algorithm, to preserve the intended properties of the original network (e.g., adversarial training [Balunovic and Vechev 2020; Madry et al. 2018] or QAT [Dong et al. 2019; Jacob et al. 2018]). Algorithm 3 presents the algorithm of QEF-Ensemble. It is initialized with a floating-point network N, training algorithm $\mathcal{T}$, network architecture $\mathcal{N}$, dataset $\mathcal{D}$, quantization scheme (S, b), and ensemble size K. It constructs an ensemble by training $K - 1$ additional networks (Line 4), using the same training algorithm but with different random seeds.

It creates the quantized versions of all networks (Line 5) and computes their corresponding `QEF` bounds for all classes (Line 7). Its inference takes an input x and iterates over the quantized networks, starting with the one corresponding to the original network N (Line 9). For each quantized network N_b^i, it passes x through the network (Line 10) and obtains the predicted class c (Line 11). Then, it checks if the classification confidence of c is greater than its corresponding `QEF` bound (Line 12). If so, it is *guaranteed* that N^i classifies x as c. Thus, c is returned along with a flag `True` indicating that x is classification consistent. Otherwise, `QEF-Ensemble` continues to the next quantized network. If none of the quantized networks meets its `QEF` bound, `QEF-Ensemble` falls back to the predicted class of the original network's quantized network along with a flag `False`, indicating that N may not necessarily classify x as c (Line 13). For example, assume a network $N^1 = N$, trained to be adversarially robust, and two additional networks N^2 and N^3, trained by the same algorithm to be adversarially robust. Given $b = 8$, we denote their quantized networks as N^1_{INT8}, N^2_{INT8}, and N^3_{INT8}. Assume their `QEF` bounds are $d[1][c] = 2$, $d[2][c] = 3$, and $d[3][c] = 1.5$. Given an input x_0, `QEF-Ensemble` passes it through N^1_{INT8}, which classifies it as c. Assume the classification confidence is $z_{L,c} - \max_{c' \neq c} z_{L,c'} = 0.5$. Since this value is below the bound $d[1][c]$, the classification c is not guaranteed to be consistent, as it lies close to the decision boundary of this classifier and may be changed by quantization. `QEF-Ensemble` thus passes x_0 through N^2_{INT8}. Assume the confidence is 3.5, which is over the bound $d[2][c] = 3$. Therefore, the classification is far from the decision boundary and is not affected by quantization errors. It is thus guaranteed to be consistent with the floating-point network N^2, and `QEF-Ensemble` returns c with a consistency guarantee.

By `QEF-Ensemble`'s operation, we get the following theorem:

Theorem 5.5. *The classification inconsistencies of* `QEF-Ensemble` *are contained in that of* N.

6 Evaluation

In this section, we evaluate the performance of `CoMPAQt` in computing the `QEF` bound and correcting the classification inconsistencies caused by the quantization. We begin by describing our experimental setup, including implementation details, the networks, and the datasets. We then describe the baselines and the experiments. Our experiments show the effectiveness of our correction mechanisms compared to the baselines as well as their robustness to out-of-distribution variations. We also present a use case over the airplane collision avoidance system using the ACAS-Xu system. We finally provide an ablation study showing the importance of `CoMPAQt`'s components.

Implementation. We implemented `CoMPAQt` in Julia 1.8.3 as a module extending MIPVerify [Tjeng et al. 2019]. Our implementation solves MILPs with Gurobi 12.0 [Gurobi Optimization, LLC 2024], whose computations are parallelized over 32 cores. The timeout to solve the main MILP (Equation (5)) is 1.5 hours and the timeout to solve the MILPs for computing the bounds (Section 5.1.4) is $T_{\text{bound}} = 10$ seconds. The threshold for the trapezoid relaxation's overapproximation error is $T_{\text{trapezoid}} = 0.1$. We ran our experiments on an Ubuntu 20.04.1 OS, using a dual AMD EPYC 7713 64-Core Processor.

Datasets and networks. We evaluate `CoMPAQt` on two tabular datasets and an image dataset:

- Adult Census [Becker and Kohavi 1996] (Adult): This dataset is used for predicting whether an individual's annual income exceeds \$50,000. Its inputs have 14 features (e.g., age and occupation) and they are labeled with yes if their income is over \$50,000 and no otherwise.
- Default of Credit Card Clients [Yeh 2016] (Credit): This dataset is used for predicting whether a client will default on payment. Its inputs have 23 features (e.g., bill amounts and age) and they are labeled with yes if they default or no otherwise.
- MNIST [LeCun and et al. 1989]: This dataset consists of grayscale images of handwritten digits. Its images consist of 28×28 pixels and they are labeled with the digit they contain.

We evaluate CoMPAQt on fully connected classifiers with two hidden layers and on a convolutional classifier with two convolutional layers followed by a fully connected layer. While the networks may seem small, we remind that CoMPAQt analyzes a *global* property, requiring to analyze every possible input. Additionally, the size of these networks is comparable to other works evaluating them [Chen et al. 2021; Kabaha and Drachsler-Cohen 2024; Reshef et al. 2024; Urban et al. 2020]. All networks use ReLU as their activation function. We consider different quantization schemes (S, b). We next describe the scale factors S, while the bit precision b is specified in the experiments. The scale factors $S = \{(s_m, s_{W,m}) \mid m \in [L]\}$ are determined by the Min-Max technique, for setting the quantization range for symmetric quantization [Nagel et al. 2021]. For each layer m, it defines the scale factor of the weights as $s_{m,W} = 2 \cdot \max(|w_{m,k}|)/(2^b - 1)$. For the layer's inputs, we consider three clipping variations: (1) W-Min-Max, which uses the actual ranges of the neurons in the layer, i.e., for each layer m, the scale factor is $s_m = \max(u_{m,k})/(2^b - 1)$. In this version, no clipping occurs, (2) α-Min-Max, a scaled variant of W-Min-Max that introduces a factor $\alpha \in [0, 1]$ to adjust the range, defined as $s_m = \alpha \cdot \max(u_{m,k})/(2^b - 1)$, allowing controlled clipping, and (3) *D-Min-Max*, which estimates the quantization range from a representative subset of the input space (we use the training set). These variations cover a wide range of quantization schemes and trade-offs between dynamic range coverage and clipping tolerance.

Baselines. CoMPAQt is novel in its ability to: (1) detect quantization classification inconsistencies for any input at inference time, (2) mitigate these inconsistencies, and (3) provide a formal guarantee on the classification consistency of the quantized network with its floating-point counterpart. To the best of our knowledge, no existing work can detect such inconsistencies, let alone mitigate them or provide a formal guarantee on the classification. Commonly used approaches such as PTQ (Post-Training Quantization) and QAT (Quantization-Aware Training) are designed to produce a quantized network that closely matches the floating-point classifier, but have no formal guarantee on being classification consistent. We compare CoMPAQt to these approaches for different quantization schemes. We implemented our baselines with the popular PyTorch quantization library[1]. For PTQ, we follow the implementation described in Nagel et al. [2021], where quantization is applied before the multiplication operations and sums are performed with higher precision (Float32 in our experiments). The weights are quantized using a signed integer scheme with W-Min-Max clipping, for each layer. For QAT, when b = INT8, we use PyTorch's `QuantStub`, `DeQuantStub`, and `torch.ao.quantization` during network training. If b = Float16, we use `torch.cuda.amp.autocast()` during the forward pass of training. This enables an automatic mixed precision: it casts operations to Float16 while maintaining numerical stability. We convert the network to Float16 using the `half()` operation. All these components are part of the standard PyTorch quantization workflow.

Evaluation metrics. To estimate the computational gain achieved by quantization and compare different schemes, we use the matrix multiplication cost, which scales quadratically with the quantization bit precision [Nagel et al. 2021]. For example, INT8 has a matrix multiplication cost of 8×8 and INT16 has a cost of 16×16, which is higher by 4x. For floating-point schemes, the multiplication is performed on the mantissa bits and it is 24×24 for Float32 and 10×10 for Float16. Since APR and QEF-Ensemble transition between different quantization schemes, we define the total matrix multiplication cost as the sum of all multiplication costs of the schemes. For example, when APR runs both INT8 and INT16 quantized networks, the total multiplication cost is $8 \times 8 + 16 \times 16$. We define the effective bit precision b_d (effective bits, for short) as the square root of the total matrix multiplication cost. In this example, $b_d = \sqrt{8 \times 8 + 16 \times 16} \approx 17.8$.

[1]https://pytorch.org/docs/stable/quantization.html

Table 1. QEF bounds over PTQ-quantized networks ($TC\%$ is classification consistency on the test set, α is 0.8).

Dataset	Network	Clipping	INT8			INT12			INT16		
			$T[h]$	d	$TC\%$	$T[h]$	d	$TC\%$	$T[h]$	d	$TC\%$
Adult	2×50	W-Min-Max	1.4	1.47	64.05	1.5	0.11	96.27	1.5	0.007	99.67
Adult	2×50	α-Min-Max	1.5	1.43	63.34	1.5	0.12	96.06	1.4	0.015	99.44
Adult	2×50	D-Min-Max	1.3	1.61	64.35	1.5	0.12	96.01	1.5	0.011	99.56
Adult	Conv	W-Min-Max	0.9	1.23	59.59	1.5	0.11	96.60	1.5	0.007	99.77
Adult	Conv	D-Min-Max	1.5	1.67	59.64	1.5	0.12	96.49	1.3	0.014	99.55
Credit	2×50	W-Min-Max	1.2	0.93	80.75	1.5	0.08	99.04	1.5	0.005	99.9
Credit	2×50	α-Min-Max	1.5	1.14	76.93	0.57	0.29	94.44	0.46	0.23	95.67
Credit	Conv	W-Min-Max	1.5	0.73	75.3	1.5	0.05	98.74	1.5	0.003	99.93
Credit	Conv	D-Min-Max	1.3	0.73	75.7	1.3	0.06	98.41	1.5	0.006	99.83
MNIST	2×50	W-Min-Max	0.4	1.85	13.95	0.68	0.16	90.98	1.46	0.012	99.42
MNIST	2×50	D-Min-Max	0.5	1.96	7.10	1.02	0.16	87.46	1.12	0.019	98.44
MNIST	Conv	W-Min-Max	0.77	6.80	15.0	0.96	0.44	94.86	1.5	0.027	99.62

Table 2. QEF bounds over QAT-quantized networks ($TC\%$ is classification consistency on the test set, α is 0.8).

Dataset	Network	Clipping	INT8			Float16		
			$T[h]$	d	$TC\%$	$T[h]$	d	$TC\%$
Adult	2×50	W-Min-Max	1.5	1.25	66.06	1.5	0.17	94.96
Adult	2×50	α-Min-Max	1.5	1.34	63.34	1.3	0.20	93.92
Adult	2×50	D-Min-Max	0.9	1.87	52.05	1.5	0.53	82.78
Credit	Conv	W-Min-Max	1.3	0.44	95.53	1.2	0.08	99.0
Credit	Conv	D-Min-Max	1.5	0.59	93.18	0.5	0.29	96.3
Adult	Conv	W-Min-Max	1.2	0.62	81.38	1.5	0.12	95.57
Adult	Conv	D-Min-Max	1.1	1.02	71.45	1.5	0.15	94.64

CoMPAQt-QEF. We begin by studying the effectiveness of CoMPAQt-QEF. We evaluate it over PTQ- and QAT-quantized networks. For PTQ, we evaluate three quantization precision levels: INT8, INT12, and INT16. For QAT, we evaluate two quantization precision levels: INT8 and Float16. For the scale factors, we consider W-Min-Max, $\alpha = 0.8$-Min-Max, and D-Min-Max. For each quantization approach and scheme, we run CoMPAQt-QEF to compute the QEF bounds of all classes. Table 1 and Table 2 report the average computed QEF bounds d across all classes, and their average computation time T in hours. Table 1 presents the results for the PTQ networks, and Table 2 presents the results for the QAT networks. To demonstrate the relevance of these bounds to the dataset's inputs, the tables also include the ratio of test set inputs whose classification confidence is above the corresponding QEF bounds $TC\%$ (i.e., the consistent classification rate). Table 1 indicates that for INT8, CoMPAQt-QEF completes in 1.1 hours and its bounds guarantee classification consistency for 54.7% of the test set inputs. For INT12, CoMPAQt-QEF completes in 1.72 hours and its bounds guarantee consistency for 90.4% of the test set. For INT16, CoMPAQt-QEF completes in 1.4 hours and its bounds guarantee consistency for 99.2% of the test set. Table 2 shows that for INT8, CoMPAQt-QEF completes in 1.3 hours and its bounds guarantee consistency for 74.7% of the test set. For Float16, CoMPAQt-QEF completes in 1.3 hours and its bounds guarantee consistency for 93.9% of the test set.

Table 3. APR vs. PTQ (b_d is effective bits, $TC\%$ is classification consistency on the test set, α is 0.8).

Dataset	Network	Clipping	PTQ Consistency $b_d = 8$	$b_d = 12$	$b_d = 16$	APR (ours) effective bits **TC = 100% guaranteed**		
			TC^{INT8} [%]	TC^{INT12} [%]	TC^{INT16} [%]	b_d^{INT8}	b_d^{INT12}	b_d^{INT16}
Adult	2×50	W-Min-Max	98.69	99.94	99.99	10.91	12.43	16.05
Adult	2×50	α-Min-Max	98.79	99.95	100	11.02	12.51	16.09
Adult	2×50	D-Min-Max	98.76	99.94	100	11.07	12.57	16.07
Adult	Conv	W-Min-Max	96.97	99.87	99.99	11.32	12.49	16.07
Adult	Conv	D-Min-Max	96.84	99.88	99.99	11.23	12.48	16.07
Credit	2×50	W-Min-Max	99.82	99.99	100	9.64	12.11	16.01
Credit	2×50	α-Min-Max	99.83	99.99	99.9	10.82	13.11	16.75
Credit	Conv	W-Min-Max	99.46	99.96	100	10.05	12.14	16.01
Credit	Conv	D-Min-Max	99.47	99.97	100	10.06	12.19	16.02
MNIST	2×50	W-Min-Max	98.66	99.91	99.99	14.6	13.34	16.14
MNIST	2×50	D-Min-Max	98.71	99.88	99.99	14.72	13.45	16.27
MNIST	Conv	W-Min-Max	99.03	99.92	99.99	13.9	12.58	16.06

APR. We compare APR (Algorithm 2), eliminating all classification inconsistencies, to PTQ and QAT. When comparing to PTQ, APR has the bit precision series (INT8, INT12, INT16, Float32) or the subseries starting from INT12 and INT16 (called by their lowest bit precision). When comparing to QAT, APR has the bit precisions (INT8, Float16, Float32) and its subseries starting from Float16. For each approach (PTQ/QAT) and series, we run APR over all the test set and compute the total multiplication cost. For example, if for input x and the series starting from INT8, the quantized network N_q^8's classification is consistent, the total multiplication cost is 8×8. If refinement to INT12 is required, the total cost is $8 \times 8 + 12 \times 12$. Given the total cost of all inputs in the test set, we compute the average cost and accordingly the effective bit precision b_d (the bit precision required to achieve the cost). The effective bit precision quantifies the overhead needed to achieve a *perfect classification consistency* with the floating-point network while still benefiting from quantization. Table 3 and Table 4 report the $TC\%$ of the standard PTQ and QAT, respectively, and our APR's effective bit precision b_d. Note that our APR's $TC\%$ is guaranteed to be 100%, and the baselines' b_d is the bit precision. Table 3 shows that, for the test set, APR with the series of INT8 is 100% classification consistent with the Float32 classifier and its effective bit precision is $b_d^{INT8} = 11.61$. Namely, it reduces the multiplication cost by 4.27x compared to the floating-point classifier. Compared to the PTQ INT8 counterpart, its cost is higher by 2.1x, but the PTQ baseline does not achieve perfect consistency and does not provide the user any guarantee for any input. For APR with the series of INT12, the effective bit precision is $b_d^{INT12} = 12.61$, reducing the multiplication cost by 3.62x compared to the floating-point classifier and increasing the cost by 1.1x compared to the PTQ INT12 counterpart. For APR with the series of INT16, the effective bit precision is $b_d^{INT16} = 16.13$, reducing the multiplication cost by 2.21x compared to the floating-point classifier and increasing the cost by 1.01x compared to the PTQ INT16 counterpart. Table 4 shows that the effective bit precision of APR with the series of INT8 is $b_d^{INT8} = 11.09$, reducing the floating-point multiplication cost by 4.68x and increasing the multiplication cost of the QAT INT8 counterpart by 1.92x. The effective bit precision of APR with the series of FLOAT16 is $b_d^{FLOAT16} = 11.57$, reducing the floating-point multiplication cost by 4.3x and increasing the cost of the QAT Float16 counterpart by 1.33x.

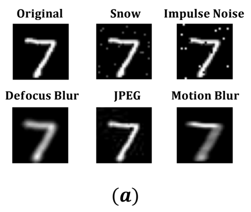
Table 4. APR vs. QAT (b_d is effective bits, $TC\%$ is classification consistency on the test set, α is 0.8).

Dataset	Network	Clipping	QAT Consistency $b_d = 8$	QAT Consistency $b_d = 10$	APR (ours) effective bit precision **TC = 100% guaranteed**	
			TC^{INT8} [%]	$TC^{FLOAT16}$ [%]	b_d^{INT8}	$b_d^{FLOAT16}$
Adult	2×50	W-Min-Max	99.35	99.99	11.04	11.35
Adult	2×50	α-Min-Max	99.49	100	11.54	11.61
Adult	2×50	D-Min-Max	99.55	99.99	13.92	14.11
Credit	Conv	W-Min-Max	99.86	100	8.79	10.28
Credit	Conv	D-Min-Max	99.80	100	9.58	11.01
Adult	Conv	W-Min-Max	98.97	99.99	11.03	11.20
Adult	Conv	D-Min-Max	98.85	99.99	11.75	11.43

Table 5. QEF-Ensemble vs. PTQ (b_d is effective bits, $TC\%$ is classification consistency on the test set). Clip. stands for Clipping, Con. for consistency, W-M. for W-Min-Max, and D-M. for D-Min-Max.

Dataset	Model	Clip.	b	PTQ Con.	Model #1		Model #2		Model #3		Model #4	
				TC [%]	TC [%]	b_d	TC [%]	b_d	TC [%]	b_d	TC [%]	b_d
Adult	2×50	W-M.	16	99.99	99.68	16	100.0	16.025	-	-	-	-
Adult	2×50	D-M.	16	99.99	98.32	16	99.94	16.02	99.993	16.03	100	16.3
Adult	2×50	W-M.	12	99.93	99.65	12	100.0	12.02	-	-	-	-
Adult	2×50	D-M.	12	99.91	95.82	12	98.15	12.24	99.90	12.35	100	12.36
Credit	Conv	W-M.	12	99.88	99.62	12	100	12.02	-	-	-	-
Credit	Conv	W-M.	8	99.48	94.83	8	97.67	8.20	98.02	8.29	99.97	8.37

QEF-Ensemble. We evaluate QEF-Ensemble (Algorithm 3) over the fully connected 2×50 network of the Adult dataset and over the convolutional network of the Credit dataset. For each, QEF-Ensemble creates an ensemble of four PTQ-quantized networks (called models). It considers the W-Min-Max or D-Min-Max clipping and the bit precisions INT8, INT12, or INT16. For each configuration, QEF-Ensemble first computes the QEF bounds of all models and classes. Then, we run QEF-Ensemble on all inputs in the test set. The total multiplication cost of QEF-Ensemble on an input x is the multiplication cost of the bit precision multiplied by the number of models used until QEF-Ensemble returns the classification. For example, for INT12, given an input x, if the first model returns a consistent classification, its cost is 12×12 and its effective bit precision is $b_d^{INT12} = 12$. Generally, if x passes through m models, its multiplication cost is $m \times 12 \times 12$, and its effective bit precision is $b_d^{INT12} = \sqrt{m \times 12 \times 12}$. Table 5 reports the classification consistency $TC\%$ of the standard PTQ and of QEF-Ensemble. For QEF-Ensemble, it also reports the effective bit precision b_d. The table details the accumulated $TC\%$ and b_d for each model. Namely, for Model #1, the table shows the $TC\%$ and b_d, over all inputs in the test set, after passing them only through Model #1. For Model #2, the table shows the $TC\%$ and b_d, over all inputs in the test set, where inputs that Model #1 could not guarantee consistency are passed through Model #2. For Model #3, the table shows the $TC\%$ and b_d, over all inputs in the test set, where inputs that Model #2 could not guarantee consistency are passed through Model #3. Similarly, for Model #4. The symbol "-"



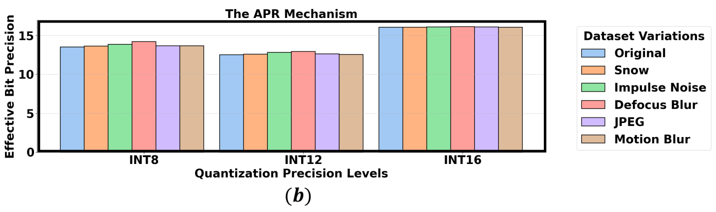


Fig. 5. CoMPAQt+APR's effective bit precision over out-of-distribution datasets.

denotes cases where a model is not invoked for any input in the test set (i.e., perfect consistency is obtained with fewer models). For the 2×50 classifier, the results show that QEF-Ensemble with bit precision INT16 or INT12 achieves *perfect consistency* with the floating-point classifier and the effective bit precision is $b_d^{INT16} = 16.16$ and $b_d^{INT12} = 12.19$ (on the test set). This reduces the multiplication cost of the floating-point classifier by 2.2x and 3.87x, respectively. It increases the cost of the PTQ baseline counterparts by only 1.02x and 1.03x, respectively. The PTQ baseline's classification consistency is 99.95% on average. However, QEF-Ensemble returns an *indication to the user* that all its classifications are consistent with the floating-point classifier, whereas the PTQ baseline cannot guarantee consistency for any input. For the convolutional classifier, QEF-Ensemble with bit precision INT12 achieves perfect consistency with the floating-point classifier and its effective bit precision is $b_d^{INT12} = 12.02$ (on the test set). This reduces the multiplication cost of the floating-point classifier by 3.98x and increases the cost of its PTQ baseline counterpart by only 1.003x. For the bit precision INT8, QEF-Ensemble obtains a consistency of 99.97%, improving the consistency of a single model by 5.14% and outperforming the PTQ baseline's consistency. While QEF-Ensemble does not achieve perfect consistency in this case, it still notifies the user when consistency cannot be guaranteed, unlike the PTQ baseline.

CoMPAQt *on out-of-distribution inputs.* We study the effective bit precision of CoMPAQt with APR on out-of-distribution dataset variations. We remind that APR guarantees perfect classification consistency for every input, including out-of-distribution inputs. We focus on common corruptions and perturbations [Hendrycks and Dietterich 2019], including snow, impulse noise, defocus blur, JPEG compression, and motion blur, exemplified in Figure 5(a). For each, we generate a dataset consisting of the perturbation of every input in the MNIST test set. We run APR for the convolutional MNIST classifier, the bit precision series (INT8, INT12, INT16, Float32) or its subseries (similarly to the experiments of Table 3), and with PTQ and the W-Min-Max clipping. Figure 5(b) shows the effective bit precision of APR. Results show that the effective bit precision of out-of-distribution datasets is similar to that of the test set (Table 3): $b_d^{INT8} = 13.75 \pm 0.22$, $b_d^{INT12} = 12.67 \pm 0.15$, and $b_d^{INT16} = 16.07 \pm 0.023$. This demonstrates the efficiency of APR in guaranteeing floating-point classifications with significantly lower multiplication cost for every input.

Case study: airplane collision avoidance. We demonstrate the importance of guaranteed classification consistency by simulating a navigation-based system. This system implements airplane collision avoidance, and it relies on networks trained on the ACAS-Xu dataset [Julian et al. 2018]. Its objective is to take navigation decisions for an airplane (Ownship) that prevent collisions with another airplane (Intruder). It is equipped with five neural network classifiers, each trained on five input features: Ownship's velocity, Intruder's velocity, their distance, the angle to Intruder, and Ownship's heading angle. The networks classify each input into one of five advisory actions: clear-of-conflict, strong-right, weak-right, strong-left, or weak-left. We use these networks to train five fully connected network classifiers with two hidden layers, each with 25 neurons. We focus

Algorithm 4: Continuous APR

```
1 Function Continuous-APR-Inference($x$)
      Input: An input $x$.
      Output: A classification consistent with $N$ minimizing the computational cost.
      Parameter : A persistent variable $next\_b$, initialized to 1.
2     foreach $i \in \{next\_b, \dots, K+1\}$ do
3         $z_L \leftarrow i \leq K ?\ N_q^{b_i}(x)\ :\ N(x)$          // Quantized or floating-point network
4         $c \leftarrow \arg\max z_L$                                // Predicted class
5         if $i = K+1$ or $z_{L,c} - \max_{c' \neq c} z_{L,c'} > d[i][c]$ then     // QEF bound is met
6             $next\_b \leftarrow i$
7             foreach $j \in [i-1]$ do
8                 if $z_{L,c} - \max_{c' \neq c} z_{L,c'} > d[j][c]$ then $next\_b \leftarrow j$; break
9             return $c$
```

on PTQ quantization and consider four bit precisions: INT4, INT8, INT12, and INT16. We adapt APR to a continuous setting, where consecutive inputs are very similar and thus expected to have similar classification confidence. Compared to APR, Continuous-APR (Algorithm 4) stores the bit precision that enabled to provide classification consistent with the floating-point in *next_b*, so the next iteration begins from this bit precision. To enable lowering the bit precision, it heuristically decreases *next_b* to the lowest precision whose QEF bound is lower than the current input's confidence. We run 10,000 simulations of airplane dynamics, using the ACAS-Xu closed-loop simulation falsification benchmark[2]. Each simulation begins by random coordinates and velocities for Ownship and Intruder and it runs for 300 seconds, invoking Continuous-APR with one classifier per second (the classifier is determined by the previous classifier's result). Figure 6 provides a visualization of one simulation, where Figure 6(a) shows the Ownship O and the Intruder I at time zero. Figures 6(b-d) illustrate the dynamics of Ownship navigating to avoid collision when using the floating-point networks (serving as our baseline) at time steps 50, 100, and 150. Figure 6(e) shows the paths taken by Ownship when using CoMPAQt with Continuous-APR. Figures 6(f)–(i) show the paths taken by Ownship using the PTQ baselines: INT4, INT8, INT12, and INT16, respectively. Figure 6(e) shows that CoMPAQt successfully follows the *exact same path* of the floating-point networks with an effective bit precision of only 10.77 bits, reducing the multiplication cost by 4.96x. In particular, it calls the 8-bit classifier in 40.54% time steps, the 12-bit classifier in 51.35% time steps, the 16-bit classifier in only 8.11% time steps, and it never calls the floating-point classifier. Figure 6(f) shows that the INT4 networks follow a completely different path, bringing the airplanes dangerously close (less than 500 ft), indicating a potential collision risk. Figure 6(g) shows that the INT8 networks also follow a different path than the floating-point networks but maintain a safe navigation distance. Figure 6(h-i) show that the INT12 and INT16 networks compute the same path as the floating-point networks. However, they pose higher multiplication cost than CoMPAQt (1.25x and 2.23x, respectively), without offering any guarantee that their flight paths align with those of the floating-point networks. Over the 10,000 simulations, CoMPAQt achieves an average effective bit precision of 9.76, reducing the floating-point multiplication cost by 6.25x, while maintaining zero path differences. The average path differences for the INT16, INT12, INT8, and INT4 networks are 4,669 ft, 18,202 ft, 72,104 ft, and 1,295,527 ft, respectively – significantly far from the desired path.

[2]https://github.com/stanleybak/acasxu_closed_loop_sim.git

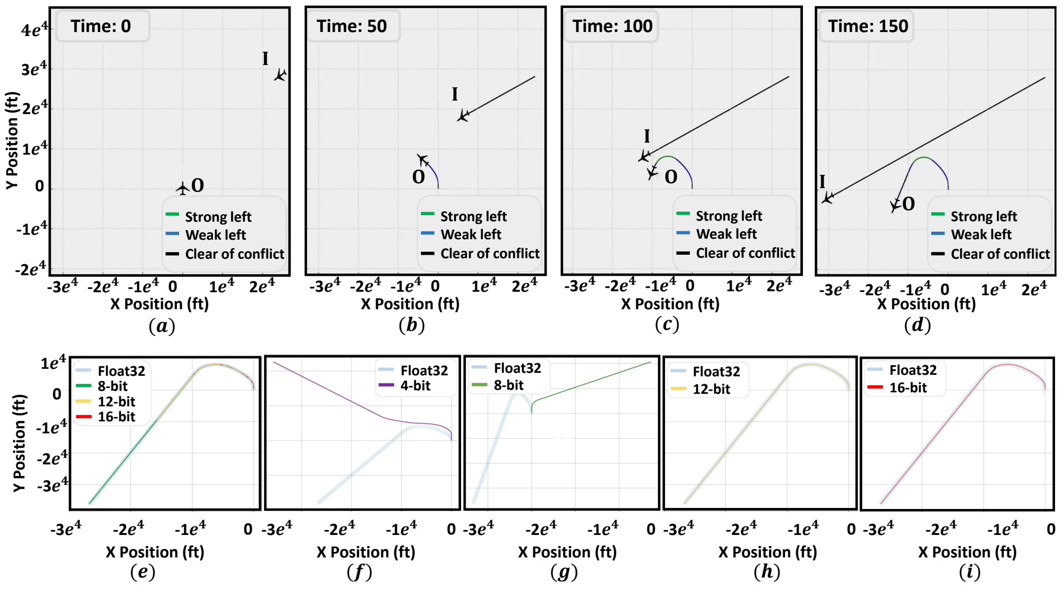


Fig. 6. ACAS-Xu case study: (a)–(d) locations of Ownship and Intruder, when relying on the floating-point networks, (e) the Ownship's path when relying on CoMPAQt+Continuous-APR (f)–(i) the Ownship's path when relying on standard PTQ-quantized networks, for different bit precisions.

Table 6. Ablation study over CoMPAQt-QEF's components.

Dataset	Network	Bit precision	MO		MO+R		MO+R+P		CoMPAQt	
			$T[h]$	d	$T[h]$	d	$T[s]$	d	$T[s]$	d
Credit	2x50	INT16	1.5	5.54	1.46	0.23	0.81	0.24	0.46	0.23
		INT12	1.5	2.50	1.5	0.32	0.87	0.26	0.57	0.29
		INT8	1.5	3.38	1.5	1.36	1.5	1.14	1.5	1.14

Ablation study. Lastly, we evaluate the effectiveness of CoMPAQt-QEF's components. We consider several variants: (1) MILP-only (MO): CoMPAQt-QEF without relations and linear relaxations, (2) MILP with relations (MO+R): CoMPAQt-QEF without linear relaxations, (3) MILP with relations and parallelogram (MO+R+P): CoMPAQt-QEF with relations and the parallelogram linear relaxation. We evaluate CoMPAQt-QEF and its variants on the 2×50 network for Credit, for the bit precisions INT16, INT12, INT8, with PTQ and the α-Min-Max clipping ($\alpha = 0.8$). We remind that the time limit is 1.5 hours. Table 6 reports the average QEF bound d over all classes and the computation time T in hours. Our results indicate that MO computes very loose bounds, which are 11.89x higher than that of CoMPAQt-QEF. Its computation time is 2.29x higher. MO+R significantly accelerates the computation time, resulting in tighter bounds than MO. However, its computation times are higher by 2.26x than CoMPAQt-QEF and its bounds are 1.1x looser than CoMPAQt-QEF. MO+R+P has tighter bounds and reduces computation time. However, it is slower by 1.42x than CoMPAQt-QEF. Its bounds are slightly tighter by 1.03x than CoMPAQt-QEF since it does not rely on the trapezoid relaxation.

7 Related Work

In this section, we discuss the closest related work to ours.

Quantized neural networks and their analysis. Several works propose post-training quantization schemes for neural networks targeting either network weights or activation outputs [Banner et al. 2019; Hubara et al. 2021; Zhang et al. 2023]. Others introduce training schemes that take into account quantization during the training process [Jacob et al. 2018; Wang et al. 2018; Zhuang et al. 2018]. Several works propose verifiers for quantized networks. Some works analyze their local robustness against adversarial attacks [Huang et al. 2024; Lin et al. 2021; Yang et al. 2024; Zhang et al. 2022]. Other work expedites this analysis for different quantization schemes by proof transfer [Ugare et al. 2022]. Another work offers probabilistic guarantees over the quantization noise [Zhang et al. 2023]. In contrast, CoMPAQt provides a deterministic formal quantization guarantee for any input. Several works propose quantization schemes with error bounds, either by bounding the difference between the output of the floating-point classifier and the output of the quantized classifier or by guaranteeing a specific round-off error bound [Ben Khalifa and Martel 2024; Lohar et al. 2023]. However, unlike CoMPAQt, these approaches cannot detect or correct classification inconsistencies, and their formal guarantees are tightly coupled with their customized quantization scheme.

Multiple network analysis. CoMPAQt computes the QEF bound by analyzing two networks. Several works analyze multiple networks or network copies. Some works offer differential verification, computing the output differences of two networks [Paulsen et al. 2020a,b]. Others focus on incremental verification, studying the differences between a network and its slightly modified version [Ugare et al. 2023]. Some works propose proof transfer between similar networks to verify local robustness [Ugare et al. 2022]. Global robustness verifiers compare the outputs of two network copies, for an input and its perturbed version [Kabaha and Drachsler-Cohen 2024; Wang et al. 2022a,b].

8 Conclusion and Future Work

We present CoMPAQt, a system providing a formal guarantee on the classification inconsistencies caused by a quantization scheme to a neural network. CoMPAQt overapproximates all classification inconsistencies with the QEF bound. It computes this bound by relying on a novel MILP encoding for quantization. To scale, it relies on linear relaxations and relations between the quantized computations to their floating-point ones. We introduce two correction mechanisms that, given the QEF bounds, identify inputs introducing classification inconsistencies at inference and mitigate them. Our first mechanism guarantees returning a classification consistent with the floating-point network, by considering networks with increasing bit precision. Our second mechanism mitigates classification inconsistencies with an ensemble of quantized networks. We evaluate CoMPAQt on tabular datasets, MNIST, and ACAS-Xu. Our results show that CoMPAQt and our correction mechanisms provide a guarantee on all inputs while reducing computational cost by 3.9x.

An interesting direction for future work is to rely on the QEF bounds to guide the selection of quantization schemes. For example, one could invoke CoMPAQt to compute the QEF bounds of different schemes and adapt the schemes with the goal of lowering the bounds. Such an extension could further motivate research on incremental QEF analysis to reduce the computation time. Another promising direction is to extend CoMPAQt to quantized networks that may exhibit inaccuracy or non-determinism due to hardware-induced effects such as noise, rounding imprecision, or bit-flip faults. Previous work has shown that such hardware-level effects can be effectively encoded as MILP [He et al. 2019; Schwan et al. 2023; Zhang et al. 2025]. Such an extension could enhance CoMPAQt's applicability in real-world safety-critical deployments.

Acknowledgements

We thank the anonymous reviewers for their feedback. This research was supported by the Israel Science Foundation (grant No. 2607/25).

Data-Availability Statement

Our code is available at https://github.com/ananmkabaha/CoMPAQt.git.

References

Mislav Balunovic and Martin T. Vechev. 2020. Adversarial Training and Provable Defenses: Bridging the Gap. *In ICLR* (2020). https://openreview.net/forum?id=SJxSDxrKDr

Debangshu Banerjee, Changming Xu, and Gagandeep Singh. 2024. Input-Relational Verification of Deep Neural Networks. *In PLDI* (2024). doi:10.1145/3656377

Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. *In NeurIPS* (2019). https://proceedings.neurips.cc/paper/2019/hash/c0a62e133894cdce435bcb4a5df1db2d-Abstract.html

Barry Becker and Ronny Kohavi. 1996. Adult. UCI Machine Learning Repository. https://doi.org/10.24432/C5XW20

Dorra Ben Khalifa and Matthieu Martel. 2024. Efficient Implementation of Neural Networks Usual Layers on Fixed-Point Architecture. *In LCTES* (2024). doi:10.1145/3652032.3657578

Arun Chauhan, Utsav Tiwari, and Vikram N. R. 2023. Post Training Mixed Precision Quantization of Neural Networks using First-Order Information. *In IEEE/CVF Workshops* (2023). doi:10.1109/ICCVW60793.2023.00144

Yizheng Chen, Shiqi Wang, Yue Qin, Xiaojing Liao, Suman Jana, and David A. Wagner. 2021. Learning Security Classifiers with Verified Global Robustness Properties. *In CCS* (2021). doi:10.1145/3460120.3484776

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2019. HAWQ: Hessian AWare Quantization of Neural Networks With Mixed-Precision. *In ICCV* (2019). doi:10.1109/ICCV.2019.00038

Rüdiger Ehlers. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *ATVA*, Deepak D'Souza and K. Narayan Kumar (Eds.), Vol. 10482. Springer, 269–286. doi:10.1007/978-3-319-68167-2_19

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual. https://www.gurobi.com

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *In CVPR* (2016). doi:10.1109/CVPR.2016.90

Zichang He, Weilong Cui, Chunfeng Cui, Timothy Sherwood, and Zheng Zhang. 2019. Efficient Uncertainty Modeling for System Design via Mixed Integer Programming. *In ICCAD* (2019). doi:10.1109/ICCAD45719.2019.8942139

Dan Hendrycks and Thomas G. Dietterich. 2019. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. *In ICLR* (2019). https://openreview.net/forum?id=HJz6tiCqYm

Qiang Hu, Yuejun Guo, Maxime Cordy, Xiaofei Xie, Wei Ma, Mike Papadakis, and Yves Le Traon. 2022. Characterizing and Understanding the Behavior of Quantized Models for Reliable Deployment. *In CoRR abs/2204.04220* (2022). doi:10.48550/ARXIV.2204.04220

Pei Huang, Haoze Wu, Yuting Yang, Ieva Daukantas, Min Wu, Yedi Zhang, and Clark W. Barrett. 2024. Towards Efficient Verification of Quantized Neural Networks. *In AAAI* (2024). doi:10.1609/AAAI.V38I19.30108

Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. 2021. Accurate Post Training Quantization With Small Calibration Sets. *In ICML* (2021). http://proceedings.mlr.press/v139/hubara21a.html

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *In CVPR* (2018). doi:10.1109/CVPR.2018.00286

Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. 2018. Deep Neural Network Compression for Aircraft Collision Avoidance Systems. *abs/1810.04240* (2018). http://arxiv.org/abs/1810.04240

Anan Kabaha and Dana Drachsler-Cohen. 2024. Verification of Neural Networks' Global Robustness. *In OOPSLA1* (2024). doi:10.1145/3649847

Anan Kabaha and Dana Drachsler-Cohen. 2025. Guarding the Privacy of Label-Only Access to Neural Network Classifiers via iDP Verification. *In OOPSLA1* (2025). doi:10.1145/3720480

Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. *In CAV* (2017). doi:10.1007/978-3-319-63387-9_5

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *In Proc. IEEE* (1998). doi:10.1109/5.726791

Yann LeCun and et al. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *In Neural Comput* (1989). doi:10.1162/NECO.1989.1.4.541

En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. 2019. Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing. *In CoRR abs/1910.05316* (2019). http://arxiv.org/abs/1910.05316

Haowen Lin, Jian Lou, Li Xiong, and Cyrus Shahabi. 2021. Integer-arithmetic-only Certified Robustness for Quantized Neural Networks. *In ICCV* (2021). doi:10.1109/ICCV48922.2021.00773

Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. MCUNet: Tiny Deep Learning on IoT Devices. *In NeurIPS* (2020). https://proceedings.neurips.cc/paper/2020/hash/86c51678350f656dcc7f490a43946ee5-Abstract.html

Shiya Liu, Dong Sam Ha, Fangyang Shen, and Yang Yi. 2021. Efficient neural networks for edge devices. *In Computers and Electrical Engineering* 92 (2021), 107121. doi:10.1016/J.COMPELECENG.2021.107121

Debasmita Lohar, Clothilde Jeangoudoux, Anastasia Volkova, and Eva Darulova. 2023. Sound Mixed Fixed-Point Quantization of Neural Networks. *In ACM Trans. Embed. Comput. Syst.* (2023). doi:10.1145/3609118

Tao Luo, Weng-Fai Wong, Rick Siow Mong Goh, Anh Tuan Do, Zhixian Chen, Haizhou Li, Wenyu Jiang, and Weiyun Yau. 2023. Achieving Green AI with Energy-Efficient Deep Learning Using Neuromorphic Computing. *In ACM* (2023). doi:10.1145/3588591

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. *In ICLR* (2018). https://openreview.net/forum?id=rJzIBfZAb

Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2022. PRIMA: general and precise neural network certification via scalable convex hull approximations. *In POPL* (2022). doi:10.1145/3498704

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. 2021. A White Paper on Neural Network Quantization. *In CoRR abs/2106.08295* (2021). https://arxiv.org/abs/2106.08295

Tianyu Pang, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. 2019. Improving Adversarial Robustness via Promoting Ensemble Diversity. *In ICML* (2019). http://proceedings.mlr.press/v97/pang19a.html

Brandon Paulsen, Jingbo Wang, and Chao Wang. 2020a. ReluDiff: differential verification of deep neural networks. *In ICSE* (2020). doi:10.1145/3377811.3380337

Brandon Paulsen, Jingbo Wang, Jiawei Wang, and Chao Wang. 2020b. NEURODIFF: Scalable Differential Verification of Neural Networks using Fine-Grained Approximation. *In ASE* (2020). doi:10.1145/3324884.3416560

Roie Reshef, Anan Kabaha, Olga Seleznova, and Dana Drachsler-Cohen. 2024. Verification of Neural Networks' Local Differential Classification Privacy. *In VMCAI* (2024). doi:10.1007/978-3-031-50521-8_5

Roland Schwan, Colin N. Jones, and Daniel Kuhn. 2023. Stability Verification of Neural Network Controllers Using Mixed-Integer Programming. *In IEEE Trans. Autom. Control* (2023). doi:10.1109/TAC.2023.3283213

Haihao Shen, Naveen Mellempudi, Xin He, Qun Gao, Chang Wang, and Mengni Wang. 2024. Efficient Post-training Quantization with FP8 Formats. *In MLSys* (2024). https://proceedings.mlsys.org/paper_files/paper/2024/hash/dea9b4b6f55ae611c54065d6fc750755-Abstract-Conference.html

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019a. An abstract domain for certifying neural networks. *In POPL* (2019). doi:10.1145/3290354

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019b. Boosting Robustness Certification of Neural Networks. *In ICLR* (2019). https://openreview.net/forum?id=HJgeEh09KQ

Thilo Strauss, Markus Hanselmann, Andrej Junginger, and Holger Ulmer. 2017. Ensemble Methods as a Defense to Adversarial Perturbations Against Deep Neural Networks. *In abs/1709.03423* (2017). http://arxiv.org/abs/1709.03423

Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *In ICML* (2019). http://proceedings.mlr.press/v97/tan19a.html

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. 2019. Evaluating robustness of neural networks with mixed integer programming. *In ICLR* (2019). https://openreview.net/forum?id=HyGIdiRqtm

Shubham Ugare, Debangshu Banerjee, Sasa Misailovic, and Gagandeep Singh. 2023. Incremental Verification of Neural Networks. *In PLDI* (2023). doi:10.1145/3591299

Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. 2022. Proof transfer for fast certification of multiple approximate neural networks. *In OOPSLA1* (2022). doi:10.1145/3527319

Caterina Urban, Maria Christakis, Valentin Wüstholz, and Fuyuan Zhang. 2020. Perfectly parallel fairness certification of neural networks. *In OOPSLA* (2020). doi:10.1145/3428253

Robert J. Vanderbei. 1996. Linear Programming: Foundations and Extensions. (1996). doi:10.1007/978-3-030-39415-8

Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. 2018. Training Deep Neural Networks with 8-bit Floating Point Numbers. *In NeurIPS* (2018). https://proceedings.neurips.cc/paper/2018/hash/335d3d1cd7ef05ec77714a215134914c-Abstract.html

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification. *In NeurIPS* (2021). https://proceedings.neurips.cc/paper/2021/hash/fac7fead96dafceaf80c1daffeae82a4-Abstract.html

Zhilu Wang, Chao Huang, and Qi Zhu. 2022a. Efficient Global Robustness Certification of Neural Networks via Interleaving Twin-Network Encoding. *In DATE 2022* (2022). doi:10.24963/IJCAI.2023/727

Zhilu Wang, Yixuan Wang, Feisi Fu, Ruochen Jiao, Chao Huang, Wenchao Li, and Qi Zhu. 2022b. A Tool for Neural Network Global Robustness Certification and Training. *In https://doi.org/10.48550/arXiv.2208.07289 2022* (2022). doi:10.48550/ARXIV.2208.07289

Yuchen Yang, Shubham Ugare, Yifan Zhao, Gagandeep Singh, and Sasa Misailovic. 2024. ARQ: A Mixed-Precision Quantization Framework for Accurate and Certifiably Robust DNNs. *In abs/2410.24214* (2024). doi:10.48550/ARXIV.2410.24214

Zhewei Yao, Xiaoxia Wu, Cheng Li, Stephen Youn, and Yuxiong He. 2024. Exploring Post-training Quantization in LLMs from Comprehensive Study to Low Rank Compensation. *In AAAI* (2024). doi:10.1609/AAAI.V38I17.29908

I-Cheng Yeh. 2016. Default of credit card clients. UCI Machine Learning Repository. https://doi.org/10.24432/C55S3H

Zhihang Yuan, Jiawei Liu, Jiaxiang Wu, Dawei Yang, Qiang Wu, Guangyu Sun, Wenyu Liu, Xinggang Wang, and Bingzhe Wu. 2023. Benchmarking the Reliability of Post-training Quantization: a Particular Focus on Worst-case Performance. *In The Second Workshop on New Frontiers in Adversarial Machine Learning* (2023). doi:10.48550/ARXIV.2303.13003

Jinjie Zhang, Yixuan Zhou, and Rayan Saab. 2023. Post-training Quantization for Neural Networks with Provable Guarantees. *In SIAM J. Math. Data Sci.* (2023). doi:10.1137/22M1511709

Yedi Zhang, Lei Huang, Pengfei Gao, Fu Song, Jun Sun, and Jin Song Dong. 2025. Verification of Bit-Flip Attacks against Quantized Neural Networks. *In OOPSLA1* (2025). doi:10.1145/3720471

Yedi Zhang, Zhe Zhao, Guangke Chen, Fu Song, Min Zhang, Taolue Chen, and Jun Sun. 2022. QVIP: An ILP-based Formal Verification Approach for Quantized Neural Networks. *In ASE* (2022). doi:10.1145/3551349.3556916

Xiaotian Zhao, Ruge Xu, and Xinfei Guo. 2023. Post-training Quantization or Quantization-aware Training? That is the Question. *In CSTIC* (2023). doi:10.1109/CSTIC58779.2023.10219214

Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian D. Reid. 2018. Towards Effective Low-Bitwidth Convolutional Neural Networks. *In CVPR* (2018). doi:10.1109/CVPR.2018.00826

Received 2025-03-26; accepted 2025-08-12