Counterexample Guided Neural Network Quantization Refinement

João Batista P. Matos Jr. , Eddie B. de Lima Filho , Iury Bessa , Edoardo Manino , Xidan Song , Lucas C. Cordeiro .

***Abstract*—Deploying Neural networks (NNs) in low-resource domains is challenging because of their high computing, memory, and power requirements. For this reason, NNs are often quantized before deployment, but such an approach degrades their accuracy. Thus, we propose the counterexample guided neural network quantization refinement (CEG4N) framework, which combines search-based quantization and equivalence checking. The former minimizes computational requirements, while the latter guarantees that the behavior of an NN does not change after quantization. We evaluate CEG4N on a diverse set of benchmarks, including large and small NNs. Our technique successfully quantizes the networks in the chosen evaluation set, while producing models with up to 163% better accuracy than state-of-the-art techniques.**

***Index Terms*—Equivalent Quantization, Neural Network Quantization, Neural Network Equivalence**

I. Introduction

Neural networks (NNs) are becoming essential in many applications such as autonomous driving [1], medicine, security, and other safety-critical domains [2]. However, current state-of-the-art NNs often require substantial computing, memory, and power resources, limiting their applicability [3]. As a result, resource-constrained systems may not be able to run complex NNs, leading to high opportunity cost for businesses.

Quantization techniques can help reduce the resource requirements of NNs [3–5] by decreasing the bit width required to represent their parameters and intermediate computation steps [4]. In general, different quantization strategies can be used. On the one hand, some studies consider only the quantization of NN weights [6–8]. On the other hand, other studies provide entire NN frameworks in integer precision, including weights, activation functions, and convolutional layers [4, 9].

Overall, the goal is compressing an NN to the smallest possible bit-width. However, doing so may affects the functional behavior of the resulting NN, making it prone to errors and loss of accuracy [5, 9]. For this reason, existing techniques usually monitor the accuracy degradation of a quantized NN (QNN) with statistical measures defined on the training set [5].

J. B. P. Matos Jr. is with the Graduate Program in Informatics, Federal University of Amazonas, Brazil (jbpmj@icomp.ufam.edu.br). E. B. Lima Filho is with TPV Technology, Brazil (eddie.filho@tpv-tech.com). I. Bessa is with the Department of Electricity, Federal University of Amazonas, Brazil (iurybessa@ufam.edu.br). E. Manino, X. Song, L. C. Cordeiro are with the Department of Computer Science, University of Manchester, United Kingdom ({lucas.cordeiro,eduardo.manino,xidan.song}@manchester.ac.uk)

Corresponding author: L. C. Cordeiro.

The work in this paper is funded by the Engineering and Physical Sciences Research Council (EPSRC) grants EP/T026995/1, EP/V000497/1, Soteria project awarded by the UK Research and Innovation for the Digital Security by Design (DSbD) Programme, and Cal-Comp Electronic by the R&D project of the Cal-Comp Institute of Technology and Innovation.

Nonetheless, statistical accuracy measures do not capture a network's vulnerability to adversarial inputs. Indeed, there may exist specific inputs for which a network's performance degrades significantly [5, 10]. Consequently, the only way to guarantee accuracy for a QNN is reformulating the problem under the notion of equivalence checking (EC) [11–13]. This property states that two NN are equivalent when they produce similar outputs for inputs in a given domain [12, 13].

This paper extends a previous work [14], which tackles the problems of NN quantization and equivalence checking in a modular fashion within the CEG4N framework, by iterating between two stages: searching for QNN candidates over a finite set of counterexamples, and verifying the QNN to either prove equivalence or generate more counterexamples. Here, we present a number of additional contributions:

- we describe the equivalent quantization (EQ) problem as a general optimization-verification iterative framework;
- we show that CEG4N works with multiple verification engines by employing neural network equivalence verification (NNEV) and satisfiability modulo theories (SMT);
- we extend the experimental evaluation of CEG4N by considering a larger set of ACAS Xu networks [15], deeper fully-connected networks for MNIST [16] and convolutional networks for CIFAR-10 [17]. Furthermore, we explore the effect of different architectures on small networks trained on the Iris [18] and Seeds [19] datasets.
- we show that CEG4N can successfully quantize NNs and produce models with up to 163% better accuracy, when compared with state-of-the-art quantization techniques.

The structure of this paper is as follows. In Section II, we introduce some preliminary information on QNNs and equivalence verification. In Section III, we give a broad survey of related work. In Section IV, we propose our improved CEG4N framework. In Section V, we present the results of our extensive experimental evaluation. In Section VI, we conclude and outline potential future work.

II. Preliminaries

A. Neural Networks

In general, NNs are non-linear multivariate functions

$$f : \mathcal{I} \subset \mathbb{R}^n \to \mathcal{O} \subset \mathbb{R}^m, \quad (1)$$

where $\mathcal{I} \subset \mathbb{R}^n$ and $\mathcal{O} \subset \mathbb{R}^m$ are the input and output domains with dimensions n and m, respectively. Internally, a NN is structured as a direct graph with a set of H hidden layers. In a feedforward NN, the neurons in each layer $h = \{0, 1, ..., H + 1\}$ are connected to those in the preceding layer $h-1$. Additionally, the neurons in the first layer $h=0$ are just a placeholder for the input of the NN, while the neurons in the last layer $h=H+1$ hold the output of function f. Fig. 1 shows a feedforward NN with $H=3$ hidden layers.

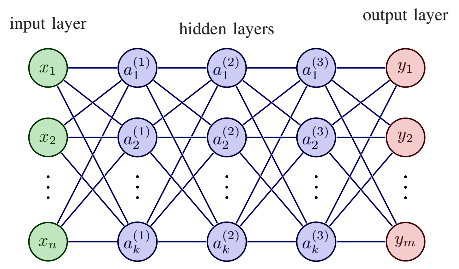

Fig. 1. A simple feed-forward NN that has three hidden layers with k neurons in each, which accepts an input of size n and produces an output of size m.

In this paper, we assume that the output $\mathbf{a^{(h)}}$ of each layer is computed by combining an affine and a non-linear transformation as follows. The non-activated and activated outputs of layer h, respectively $\mathbf{z}$ and $\mathbf{a^{(h)}}$, are:

$$\mathbf{z^{(h)}} = \mathbf{W^{(h)}} \cdot \mathbf{a^{(h-1)}} + \mathbf{b^{(h)}}, \tag{2}$$
$$\mathbf{a^{(h)}} = \sigma(\mathbf{z^{(h)}}), \tag{3}$$

where $\mathbf{W^{(h)}} \in \mathbb{R}^{m_{h-1}\times m_h}$ is weights' matrix, $\mathbf{b^{(h)}} \in \mathbb{R}^{m_h}$ is the bias vector, $\sigma : \mathbb{R}^{m_h} \to \mathbb{R}^{m_h}$ is the non-linear activation function, $\mathbf{a^{(0)}} = \mathbf{x}$ is the NN input, and the layers dimensions satisfy $n = m_0$ and $m_{H+1} = m$. The most popular activation functions σ are the rectified linear unit (ReLU), the sigmoid (Sigm), the hyperbolic tangent (TanH), and the max pooling operator [13]. While our framework is agnostic to the specific choice of σ, in our experiments we focus on ReLU:

$$\mathrm{ReLU}(\mathbf{z^{(h)}}) = \max\left\{0, \mathbf{z^{(h)}}\right\}. \tag{4}$$

Our framework support both fully-connected and convolutional layers. More specifically, (2) can be written as follows:

$$\begin{pmatrix} z_1^{(h)} \\ z_2^{(h)} \\ \vdots \\ z_{m_h}^{(h)} \end{pmatrix} = \begin{pmatrix} w_{1,0}^{(h)} & \dots & w_{1,m_{h-1}}^{(h)} \\ w_{2,0}^{(h)} & \dots & w_{2,m_{h-1}}^{(h)} \\ \vdots & \ddots & \vdots \\ w_{m_h,0}^{(h)} & \dots & w_{m_h,m_{h-1}}^{(h)} \end{pmatrix} \begin{pmatrix} a_1^{(h-1)} \\ a_2^{(h-1)} \\ \vdots \\ a_{m_{h-1}}^{(h-1)} \end{pmatrix} + \begin{pmatrix} b_1^{(h)} \\ b_2^{(h)} \\ \vdots \\ b_{m_h}^{(h)} \end{pmatrix},$$

where the matrix $\mathbf{W^{(h)}}$ is dense in fully-connected layers and sparse in convolutional ones [20].

B. Quantization

Quantization is the process of constraining high-precision values (e.g., single-precision floating-point values) to a finite range of low-precision ones (e.g., integers) [5, 21]. The quantization quality is usually determined by a scalar n (the available number of bits) that defines a finite range's lower and upper bounds. Let us define quantization as a mapping function $\mathcal{Q}_n : \mathbb{R}^{m\times p} \to \mathbb{I}^{m\times p}$, which is formulated as

$$Q\left(A, n\right) = clip\left(\left\lfloor \frac{A}{q\left(A, n\right)} \right\rceil, -2^{n-1}, 2^{n-1}-1\right), \tag{5}$$

where $A \in \mathbb{R}^{m\times\ p}$ denotes a high-precision (i.e., single scalar, vector, or matrix) value; n is the number of bits used for quantization; $q(A,n)$ means a function that computes the scaling factor for A concerning a number of bits n; *clip* denotes a clipping function that ensures the values being mapped by the quantization function are bounded by some upper lower and upper values; $\lfloor\cdot\rceil$ denotes rounding to the nearest integer; and -2^{n-1} and $2^{n-1}-1$ indicate lower and upper bounds of the clipping function, respectively. Defining a scaling factor (see Eq. 6) is an important aspect when dealing with uniform quantization [9, 22]. Moreover, the original high-precision values to be quantized are floating-point ones, given that NNs are usually designed using this representation.

The scaling factor divides a given range of values A into an arbitrary number of partitions. Thus, let us define a scaling factor function $q(A,n)$, a number of bits (bit-width) n to be used for quantization, and a clipping range given by $[\alpha,\beta]$, which leads to a scaling factor defined as

$$q\left(A, n\right) = \frac{\beta - \alpha}{2^n - 1}. \tag{6}$$

We use symmetric quantization, thus the clipping values are $\beta = -\alpha = \max([|\min(A)|, |\max(A)|])$. A quantization process can produce an integer value outside the quantized range. To prevent that, an additional clipping step is necessary. A de-quantization process computes back original values as:

$$\hat{A} = q\left(A, n\right) Q\left(A, n\right). \tag{7}$$

However, both clipping and rounding cause permanent loss of information. Consequently, de-quantization can only approximate original values, i.e., $A \approx \hat{A}$.

C. Neural Network Quantization

When we quantize a NN, we can follow a number of different strategies [4, 9] Here, we will consider only the strategy of storing all NN weights in quantized format to reduce the memory requirements [6–8]. At inference time, we assume that the weights are de-quantized and all the operations in the NN are executed in floating point.

Consider a feedforward NN f as defined in (2) and (3). Call $\mathcal{N}_f$ the set whose elements $n^{(h)} \in \mathcal{N}_f$ are the bit widths for each layer h in f. Given the de-quantization process in (7) and the non-activated output in (2), we can describe the non-activated output $\hat{\mathbf{z}}^{\mathbf{(h)}}$ of a quantized layer h as:

$$\begin{aligned}\hat{\mathbf{z}}^{\mathbf{(h)}} = {} & q\left(\mathbf{W^{(h)}}, n^{(h)}\right) Q\left(\mathbf{W^{(h)}}, n^{(h)}\right) \cdot \mathbf{a^{(h-1)}} \\ & + q\left(\mathbf{b^{(h)}}, n^{(h)}\right) Q\left(\mathbf{b^{(h)}}, n^{(h)}\right).\end{aligned} \tag{8}$$

D. Neural Network Equivalence

Let $f : \mathcal{I} \to \mathcal{O}$ and $f' : \mathcal{I} \to \mathcal{O}$ be two arbitrary NNs, where $\mathcal{I} \in \mathbb{R}^n$ and $\mathcal{O} \in \mathbb{R}^m$ are their common input and output spaces, respectively. Currently, the literature reports the following definitions of equivalence [11–13, 23, 24].

*Definition 2.1 (*Top-Equivalence*):* Consider two NNs $f : \mathcal{I} \to \mathcal{O}$ and $f' : \mathcal{I} \to \mathcal{O}$. Then, f and f' are Top-1-equivalent, i.e., $f \equiv f'$, if and only if the following holds:

$$\forall\ x \in \mathcal{I}, f(x) = f'(x). \tag{9}$$

Definition 2.2 (ϵ-Equivalence): Consider two NNs $f : \mathcal{I} \to \mathcal{O}$ and $f' : \mathcal{I} \to \mathcal{O}$, and an $\epsilon > 0$. Then, f and f' are ϵ-equivalent, i.e., $f \backsim_{p,\epsilon} f'$, if and only if the following holds:

$$\forall\ x \in \mathcal{I}, ||f(x) - f'(x)||_p \leq \epsilon. \tag{10}$$

Definition 2.1 is a strict form of equivalence and imposes a hard requirement [13]. Definition 2.2, in turn, is a flexible form of equivalence [12]. As noted by Eleftheriadis *et al.*[13], *Top-Equivalence* is a true equivalence relation, that is, it is reflexive ($f \equiv f$ for any NN f), symmetric ($f \equiv f'$ iff $f' \equiv f$), and transitive ($f \equiv f'$ and $f' \equiv f''$ implies $f \equiv f''$). However, ϵ-*Equivalence* is only reflexive and symmetric.

E. Verification of equivalence properties

The goal of NNEV verification is to check if $f \backsimeq f'$. Then, we define NNEV as follows.

Definition 2.3 (Neural network verification problem): Given two NNs f and f' and an equivalence relation $\backsimeq \in \{\equiv, \backsim_\epsilon\}$, NNEV consists in checking if $f \backsimeq f'$.

This paper uses two paradigms: SMT and reachability analysis (RA). With SMT, the equivalence property and the NN model are encoded as a first-order logic formula. SMT restricts the full expressive power of first-order logic to a decidable fragment. With RA, in the form of GPE, the property to be checked and the model are encoded as linear constraints.

1) SMT Encoding: SMT formulas can capture the complex relationship between variables holding real, integer, and other data types. If it is possible to assign values to them so that a formula is evaluated as true, then it is said to be satisfiable. However, if assigning such values is impossible, the mentioned formula is considered unsatisfiable.

The following steps show how to reduce neural network equivalence (NNE) to a logical satisfiability problem: (1) encoding f into an SMT formula ϕ; (2) encoding f' into an SMT formula ϕ'; (3) encoding the relation $f \backsimeq f'$ into an SMT formula Φ, such that $f \backsimeq f'$ iff Φ is not satisfiable; and (4) checking, via SMT solver, whether Φ is satisfiable. If the latter is true, f and f' are not equivalent, and the solver provides a counterexample. Otherwise, f and f' are equivalent.

Checking NNE becomes possible by relying on the negation of $f \backsimeq f'$, i.e., by encoding it as a formula that asserts the existence of an input $x \in \mathcal{I}$ and two outputs $y, y' \in \mathcal{O}$ ($y = f(x)$ and $y' = f'(x)$) such that they do not satisfy the conditions imposed by $\backsimeq$. Indeed, we check if

$$\left(\exists x \in \mathcal{I}; y, y' \in \mathcal{O}; y = f(x) \wedge y' = f'(x) \wedge y \neq y'\right),$$

or, regarding the ϵ-*Equivalence* definition, if

$$\left(\exists x \in \mathcal{I}; y, y' \in \mathcal{O}; y = f(x) \wedge y' = f'(x) \wedge ||y - y'||_p > \epsilon\right).$$

In summary, checking whether the formula $y = f(x) \wedge y' = f'(x) \wedge y \neq y'$ is unsatisfiable can be further expressed as

$$\begin{aligned} \phi &:= y = f(x) \\ \phi' &:= y' = f'(x) \\ \Phi &:= \phi \wedge \phi' \wedge y \neq y'. \end{aligned} \tag{11}$$

Similarly, checking whether the formula $y = f(x) \wedge y' = f'(x) \wedge ||y - y'||_p > \epsilon$ is unsatisfiable can be expressed as

$$\begin{aligned} \phi &:= y = f(x) \\ \phi' &:= y' = f'(x) \\ \Phi &:= \phi \wedge \phi' \wedge ||y - y'||_p > \epsilon. \end{aligned} \tag{12}$$

In addition, the input of an NN f is a vector $\vec{x} = [x_1, ..., x_n] \in \mathbb{R}^n$, and some limitation regarding it may be necessary. This constraint can then be added as

$$\bigwedge_{j=1}^{n} x_j - r \leq x_j \leq x_j + r. \tag{13}$$

In practice, we define a limiting region between $x_j - r$ and $x_j + r$ around every point $x_j \in \mathbf{x}$, where equivalence is more likely. It works as another relaxation factor for equivalence because the associated properties should hold only for a restricted input domain. It is also corroborated by the notion that we expect the same behavior from a close neighbor of an input. In addition, it can also be linked to real conditions of a given application, such as its equivalence method and input deviation and magnitude. However, this does not mean that our technique is limited to small input ranges. Instead, it is important to choose a range that preserves the relationship between x and y and is also according to a specific application.

A possible way to use SMT is to employ a verifier based on it and then provide a suitable model. One example is the efficient SMT-based bounded model checker (ESBMC), which supports SMT solvers natively. It generates verification conditions for a given C or C++ program, i.e., its input model, encodes them using different SMT background theories (i.e., linear-integer, real arithmetic, and bit-vectors), and employs different solvers (e.g., Boolector [25] and Z3 [26]).

2) Geometric Path Enumeration (GPE) Encoding: Tran et al., 2019 [27] proposed GPE, a methodology for verifying NNs' safety properties and the verification approach used by the tool proposed by Teuber *et al.* [12], namely NNEQUIV. We briefly describe NNEQUIV and how it encodes NNs and equivalence property (EP) into a verification problem.

Definition 2.4 (Generalized Star Set [27]): A generalized start set Θ is a tuple $\langle\ c, G, P\rangle$ where $c \in \mathbb{R}^n$ is the center, $G = (g_1, ..., g_m) \in \mathbb{R}^{n \times m}$ is the generator matrix, and $P \subseteq \mathbb{R}^m$ is a polytope defining a conjunction of linear constraints. The set represented by Θ is then defined as

$$\Theta = \{x \in \mathbb{R}^n \mid \exists \alpha \in P : x = c + G\alpha\}.$$

Assume we have two NNs f and g representing piecewise linear functions. Furthermore, assume that we want to verify whether $f(x) = g(x)$ for the input domain $\mathcal{I} \equiv \langle\ c, G, P\rangle$. Since f and g are piecewise linear, there exist a tiling $\mathcal{T} \equiv \{P'\}$ of the input domain $\mathcal{I}$ such that $\mathcal{I} = \{x \in \mathbb{R}^n \mid \exists P' \in \mathcal{T} \wedge \alpha \in P' : x = c + G\alpha\}$. Moreover, for each tile $P' \in \mathcal{T}$, we require both f and g to be linear, i.e., $f(x) = c_f + G_f\alpha$ and $g(x) = c_g + G_g\alpha$ for $\alpha \in P'$, where (c_f, c_g, G_f, G_g) are specific to each tile P'. The neural networks f and g are equivalent if $c_f = c_g$ and $G_f = G_g$ for each tile P'.

The NNEQUIV's algorithm [12] computes the tiling $\mathcal{T}$ via repeated reachability analysis as follows. First, the input domain $\mathcal{I}$ is propagated through NN f, outputting a union of star sets. Then, each set is projected back onto the input space and propagated through NN g, leading to a further union of star sets whose elements, once projected onto the input domain, represent the tiles $P' \in \mathcal{T}$.

III. Related Work

A. Neural Network Quantization

There are many aspects to consider when deciding to deploy a quantization scheme [5]. For instance, if the goal is to reduce an NN's size, one can consider quantizing only its weights and biases [5, 9, 28]. However, if the goal is to reduce computation and memory requirements, one can consider quantizing weights, biases, and activation functions [9, 28].

Indeed, the quantization of activation functions can reduce computational and memory costs [9, 28]. However, it raises challenges as it usually requires a calibration step using representative data, prior to quantization, to correctly compute quantization ranges [5, 28]. Conservative approaches based on neurons' transfer functions can also be used [29], but they might lead to poorly quantized regions. In this regard, the technique proposed here aims to reduce an NN's size as it applies a method that only quantizes its weights and biases.

In fact, several studies have focused specifically on quantizing weights of NNs [6, 8, 30–33] For instance, Courbariaux *et al.* [30] proposed a method called binarized neural networks, which lowers the storage costs of an NN's weight parameters by reducing them to binary values. Other studies have also explored mixed-precision quantization techniques, which quantize NN weights while retaining activation functions in full precision. Yuan *et al.* [32] proposed EvoQ, which uses evolutionary search to achieve mixed precision quantization without access to complete training datasets. Zhou *et al.* [6] proposed the incremental network quantization, which converts a convolutional NN into a lower precision version whose weights can only be either powers of two or zero, considering weight importance to keep high accuracy. Finally, some studies store quantized weights with floating-point precision, thus facilitating integration for inference and generalization [8, 33].

B. Quantization-Aware Training

Another important aspect to consider is whether to employ a post-training quantization strategy, as we do in the present paper, or allow for some form of weight retraining. The latter may recover some of the performance lost due to the quantization and has been a very active area of research [34–36]. One of the fundamental problem is expressing the underlying optimization problem in a gradient-friendly form. In this way, the quantization objective can be included in the regular loss function during training [35].

As the size of neural networks has grown larger, more recent work attempts to quantize the training gradients too. As an example, Zhou *et al.* [37] proposed a method called DoReFa-Net for training convolutional neural networks with low bitwidth weights, activations, and gradients. During the training process, parameter gradients are quantized to low bitwidth values, allowing faster training and inference using bit convolution kernels. This approach is efficient on various hardware platforms like CPU, FPGA, ASIC, and GPU.

Alternatively, higher rates of compression can be achieved by using different quantization schemes on different regions of the input. For instance, Huang *et al.* [38] propose a dynamic quantization strategy that avoids a non-uniform usage of the available computational resources. Their technique is particularly suited to deployment on hardware accelerators.

Unfortunately, quantization-aware training may break the assumptions of the existing NNE verification tools [11–13, 39]. Thus, in this paper, we focus on post-training quantization.

C. Verification of Quantized Neural Networks

Giacobbe *et al.* [40] are the first to formally investigate the impact of quantization on NNs. They explore how quantization affects the NNs' robustness and formal verification. They propose a bit-precise SMT-solving approach for determining the satisfiability of first-order logic formulas where variables represent fixed-size bit-vectors. Their study shows that there is no simple and direct correlation or pattern between the robustness and the number of bits of a QNN. and makes several significant contributions, including revealing non-monotonicity in QNN robustness, introducing a complete verification method, and highlighting the limitations of existing approaches.

Henzinger *et al.* [41] proposed an SMT-based verification method for QNNs, using bit-vector specifications. It requires translating an NN and its safety properties into closed quantifier-free formulae over the theory of fixed-size bit-vectors. It performs verification only, focusing on robustness, while CEG4N tackles both NN quantization and verification, trying to find a more compact representation that is sound, with NN translation into formulae done by a verifier.

Mistry *et al.* [42] discusses the formal verification of QNNs implemented using fixed-point arithmetic. The authors propose a novel methodology for encoding the verification problem into a mixed-integer linear programming (MILP) problem, focusing on the bit-precise semantics of QNNs. Their results demonstrate that their MILP-based technique outperforms state-of-the-art bit-vector encodings by a significant margin.

Song *et al.* [43] proposed *QNNVerifier*, which performs SMT-based verification of QNNs. Their technique relies on fixed-point operational models for using the C language as an abstract model, which allowed operations to be encoded in their quantized form, explicitly, thus providing compatibility with SMT solvers. Although this study presents some similarities with ours, the main difference lies in the properties being verified. It checks if a QNN is invariant to adversarial inputs, while CEG4N iteratively verifies if an NN is invariant to quantization, with decoupled quantization and verification.

D. Neural Network Equivalence

Our CEG4N framework can in principle accommodate multiple equivalence verification techniques. In addition to those, we mention in Section II, all the following are viable alternatives. First, Büning *et al.* [11] defined the notion of relaxed equivalence, because exact equivalence (see Section II-D) is hard to solve. They choose to encode equivalence properties into MILP. First, the input domain is restricted to radii around a point, where equality is more likely. Then, a less strict relation is used. Finally, two NNs $\mathbf{R}$ and $\mathbf{T}$ are equivalent if the classification result of $\mathbf{R}$ is amongst the top-K largest results of $\mathbf{T}$. It was later extended by *Teuber et al.* [12].

Furthermore, Eleftheriadis *et al.* [13] proposed an SMT-based NNE checking scheme based on strict *ϵ-Equivalence*. The key differences between their work and the equivalence checking techniques we use are as follows. We also consider *Top-Equivalence*, and we encode NNs, EP, and equivalence relation either as a C program or Python code along with a structured description. More recently, Zhang *et al.* [39] introduced QEBVerif, a method that is capable of verifying the equivalence between a NN and its quantized counterpart when both the weights and the activation tensors are quantized. QEBVerif consists of differential reachability analysis (DRA) and MILP-based verification. Similar to the work by Eleftheriadis *et al.* [13], they mainly focus on *ϵ-Equivalence*, and their technique cannot be easily extended to *Top-Equivalence*.

IV. COUNTEREXAMPLE GUIDED NEURAL NETWORK QUANTIZATION REFINEMENT

This work aims to provide a methodology for creating compressed NNs that are as small as possible, from a quantization point of view. We provide the definition of EQ as follows.

Definition 4.1 (NN Quantization Vector): Let $\mathcal{N} = (n_0, \ldots, n_{H-1})$ be a vector that contains the bit width n_h for the weights of each layer h of a QNN f_q.

Definition 4.2 (Equivalent Quantization): Let f be the reference NN, $\mathcal{H} \in \mathbb{R}^n$ be a set of input instances, and $\simeq \in \{\equiv, \backsim_{p,\epsilon}\}$ an NNE relation. A vector of bit widths $\mathcal{N}$ can be used to quantize f and obtain its quantized version f^q. Thus, EQ searches for a vector $\mathcal{N}$ for which f and f^q satisfy the following equivalence constraint: $f(x) \simeq f^q(x)\ \forall\ x \in \mathcal{H}$.

From the definitions of NNE discussed in Section II-D, we preserve the equivalence between the mathematical functions f and f^q associated with the original and QNNs, respectively. In more detail, consider an NN f with H layers. As stated in Definition 4.1, its quantization assumes that there is a vector $\mathcal{N}$ whose elements n_h represent the bit width that should be used to quantize the h-th layer in f, with $h = [0, 1, 2, ..., H-1]$. In our EQ problem, we obtain a vector $\mathcal{N}$ whose elements n_h are minimized while keeping f and f^q equivalent. To obtain $\mathcal{N}$, one can apply an optimization algorithm.

A. EQ as a Minimization Problem

We consider the EQ processing of an NN as an iterative minimization problem. Specifically, each iteration is composed of two complementary sub-problems. First, we need to optimize the numbers of bits for quantization, i.e., finding a candidate vector $\mathcal{N}$ that holds all minimum bit widths. Second, we need to verify the equivalence property, i.e., checking if an NN quantized with the bit widths in $\mathcal{N}$ is equivalent to its original model. If the latter fails, we iteratively return to the minimization sub-problem with additional information. More formally, we define the first sub-problem as follows.

Optimization sub-problem o:

$$\begin{aligned} \textbf{Objective:} \quad & \mathcal{N}^o = \underset{n_o^0,\ldots,n_o^{H-1}}{\arg\min} \sum_{h \in \mathbb{N}_{h<H}} p^{(h)} * n^{(h)} \\ \textbf{s.t:} \quad & f(x) \simeq f^q(x),\ \forall\ x \in \mathcal{H}^o_{\text{CE}} \\ & n^{(h)} \geq \underline{N}\ \forall\ n^{(h)} \in \mathcal{N}^o \\ & n^{(h)} \leq \overline{N}\ \forall\ n^{(h)} \in \mathcal{N}^o \end{aligned} \tag{14}$$

Here, f is the function associated with NN $\mathcal{F}$, f^q is the quantized function associated with NN $\mathcal{F}$, and $\mathcal{H}^o_{\text{CE}}$ is a set of counterexamples that may be available at iteration o.

Consider $\underline{N}$ and $\overline{N}$ as the minimum and maximum bit widths allowed for quantization that ensure two aspects as follows. First, they provide lower and upper bounds for the quantization bit width, which guarantees correctness for the quantization process and the generation of valid quantized models. Second, they can be regarded as initialization and termination criteria, the latter if a candidate $\mathcal{N}^o$ such that $n^{(h)} = \overline{N}$ for every $n^{(h)} \in \mathcal{N}^o$ is reached. When it happens, the optimization process is stopped as no valid quantized model could be generated. In any case, if CEG4N proposes a quantization solution where $n = \overline{N}$ for every $n \in \mathcal{N}^o$, it is verified as well. Besides, if the verification process returns a counterexample, CEG4N finishes with failure. Finally, note that $\mathcal{H}^o_{\text{CE}}$ is an iterative parameter updated at each iteration o and based on the verification sub-problem.

Moreover, the function being minimized represents a weighted summation of the bit widths in the solution candidate $\mathcal{H}^o$, where $p^{(h)}$ is a constant value associated with the bit width $n^{(h)}$. These constants allow the optimization algorithm to prioritize layers based on their sizes. In our case, we have defined weights proportional to the number of neurons in a given layer h: more extensive layers, in terms of neurons, have bigger weights. This way, the optimizer searches for solutions with smaller bit widths associated with larger layers, which straightly represent their complexities.

Our search-based technique is very simple as a unique precision is adopted for an entire NN layer, i.e., we do not differentiate channels in a single convolution layer. Since it favors the combination of quantization and formal verification, we prioritize the feasibility of the overall framework.

Verification sub-problem o:

$$\Phi := \phi \wedge \phi' \wedge \neg(y \simeq y')$$

In the verification sub-problem o, we check whether $\mathcal{N}^o$ generated by the optimization sub-problem o satisfies one of the properties presented in Section II-E, depending on the chosen form of equivalence. If Φ holds for the candidate $\mathcal{N}^o$, the optimization halts and $\mathcal{N}^o$ is presented as a solution; otherwise, a new counterexample x_{CE} is generated. Then, the iteration $o+1$ starts where the iteration o stops. Consequently, the optimization sub-problem $o+1$ receives as parameter a set of $\mathcal{H}^{o+1}_{\text{CE}}$ such that $\mathcal{H}^{o+1}_{\text{CE}} = \mathcal{H}^o_{\text{CE}} \cup x_{\text{CE}}$, which is used as additional information for successful execution.

B. The CEG4N Framework Implementation

We propose the CEG4N framework, a counterexample-guided optimization approach to solve the EQ problem pre-

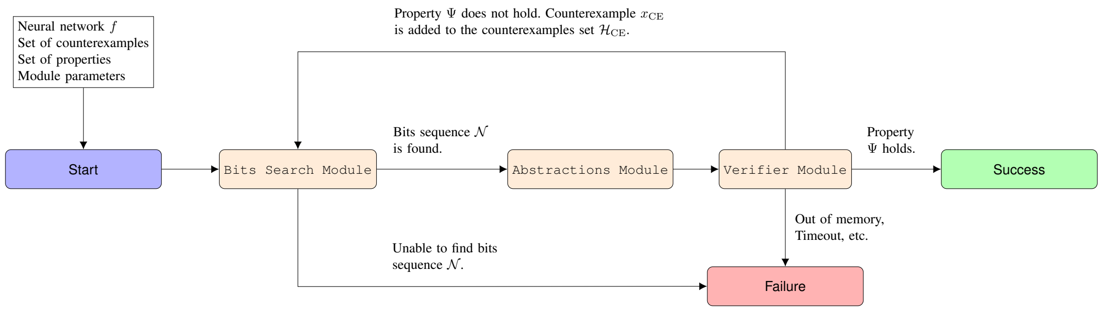
Neural network f
Set of counterexamples
Set of properties
Module parameters
Start
Bits Search Module
Bits sequence $\mathcal{N}$ is found.
Abstractions Module
Verifier Module
Property Ψ does not hold. Counterexample x_{CE} is added to the counterexamples set $\mathcal{H}_{\text{CE}}$.
Property Ψ holds.
Success
Out of memory, Timeout, etc.
Unable to find bits sequence $\mathcal{N}$.
Failure

Fig. 2. An overview of CEG4N's architecture, highlighting the relationship between main modules and their inputs and outputs.

sented here. It integrates different techniques to tackle the two sub-problems described in Section IV-A: the optimization of bit widths and the verification of NN equivalence. Indeed, CEG4N is designed to combine three modules, each characterized by a specific role in EQ. Fig. 2 illustrates its architecture.

1) Bits Search Module (BSM): The first element, namely *bits search module* (BSM), is an instance of the optimization algorithm, e.g., a genetic algorithm (GA). BSM expects three main inputs: the original NN, a set of counterexamples $\mathcal{H}_{\text{CE}}$, and a set of equivalence properties (see Equation 14). Its output is a vector of bits containing the bit width for each layer in a given NN. We can also specify lower and upper bounds to restrict the possible widths, and, depending on the chosen optimization algorithm, other parameters may also be required. For instance, if we use GAs, we may need to specify the maximum number of generations they are allowed to run. Besides, there is no limitation regarding optimization algorithms, i.e., any technique could be used. However, in this paper, we only support and describe a GA module.

2) Abstractions Module (AM): The second element, namely *abstractions module* (AM), encodes the original and QNNs into a format the third module can handle. Depending on the choice for the latter, the functional behavior of an NN is represented at different levels of abstraction. Specific examples include the open neural network exchange (ONNX) file format, which stores the architecture of an NN and its weights, and a C/C++ source file, which provides a low-level NN implementation. Generally, we encode the original neural network and its quantized counterpart separately.

At this point, it is important to explain the internal NN representations used in CEG4N. Specifically, it handles two functional versions of the same NN, at the same time:

- a version for optimization written in Python, in BSM;
- a version for verification written in C, when ESBMC is used, or Python, when NNEQUIV is employed, in VM.

Such versions are equivalent as they share the same parameters (i.e., weights, bias, and activation functions), while the major difference resides in how their operations are implemented. BSM works with an NN representation written in Python, in addition to an ONNX model. It is loaded along with its weights into the *Pytorch* framework, which implements all NN mathematical operations. The other two representations are created in AM and used in VM. If ESBMC is chosen as verifier (see Section II-E1), a functional NN version written in C/C++ and adapted to it is used. However, if NNEQUIV is chosen (see Section II-E2), a functional version written in Python is employed. The latter implements all mathematical operations directly in Python, with no additional framework.

3) Verifier Module (VM): The third and final element, which we call the *verifier module* (VM), receives as inputs the previously mentioned NN abstractions and a set of equivalence properties. Then, it checks whether the latter holds for the former. When any given property does not hold, a counterexample is provided by VM. Currently, CEG4N supports two options: 1) a bounded model checker, namely ESBMC [44, 45]), and a GPE encoder, namely NNEQUIV [12].

In summary, NN equivalence is checked using SMT or GPE. In the first, an NN is encoded into a C program that is translated into SMT by ESBMC. In the second, NNEquiv takes an NN description in Python, encodes it into star sets, and handles the result as a linear programming problem.

However, there is no essential restriction regarding verification techniques as long as counterexamples are provided. That is an interesting compromise configuration regarding any desired aspect, such as scalability, accuracy, or speed. In addition, we believe CEG4N is the first framework to combine quantization and equivalence checking, with formal guarantees for the equivalence between quantized and original NNs.

An important aspect is that spurious counterexamples may be generated in VM (see Section II-E2). To prevent them from being added to $\mathcal{H}_{\text{CE}}^{o+1}$, after each step o, we double-check them using the original and QNNs. If their predictions diverge, the respective counterexamples are discarded.

4) High-level overview of a CEG4N run: A successful CEG4N run can be summarized in the following steps:

1) CEG4N starts with NN, a set of counterexamples, the BSM's parameters, and the VM's equivalence properties;
2) BSM runs to look for a set of bit widths;
3) if none is found, CEG4N stops;
4) otherwise, AM creates NN abstractions based on the set of bit widths found by BSM;
5) VM runs with the previously generated NN abstractions and the equivalence properties specified in Step 1);
6) if these properties hold, CEG4N stops and produces a QNN based on the bit widths found in Step 2);
7) otherwise, a new set of counterexamples, consisting of elements specified in Step 1) and also new ones produced by VM in Step 4), is created;
8) the execution goes to Step 2), until a timeout is reached;

5) Discussion: We will now discuss some aspects of CEG4N. First, BSM relies on a GA module to find QNN candidates. It guarantees termination, using timeouts, but not that the most compact model will be found. Then, we need to consider VM, which, due to verification aspects, does not ensure completeness; however, it is sound. If a given property is satisfied for a QNN candidate, it will produce a counterexample as proof that it is not equivalent to its original counterpart; otherwise, this QNN candidate satisfies the specified equivalence properties. Therefore, CEG4N can find, when possible, a QNN that satisfies the equivalence requirements regarding its original counterpart.

One may also argue that encoding the quantization problem into SMT formulae could be used to replace our search-based module. However, it is clear that handling only NN verification with SMT solvers already leads to large state spaces [29], complex encoding [41], and long run times, the latter shown here, which indicates that a possible unified framework including quantization will be even more complex. Consequently, we believe that the decomposition strategy resulting from decoupling quantization and verification makes EQ tractable.

V. Experimental Evaluation

A. Description of the Benchmarks

We evaluate our methodology based on feedforward and convolutional NN classification models extracted from the literature [13, 15, 16]. We have chosen them mainly based on their popularity in previous verification studies [12, 13]. Additionally, we have included other NN models to cover a broader range of NN architectures (e.g., size and number of neurons). The chosen benchmarks are presented below.

1) ACAS Xu: This dataset is derived from eight specifications, including features, decision boundaries, and expected outputs [15]. Its features are sensor data indicating the speed and course of the monitored aircraft and the position and speed of any nearby intruder. Its NNs are expected to give appropriate navigation advisories for input sensor data. The expected outputs indicate that either the aircraft is clear of conflict or it should take soft or hard turns to avoid a collision. We have evaluated CEG4N on nine pre-trained NNs [46], each containing 6 layers and a total of 300 ReLU nodes, which were obtained from the VNN-COMP 2021 's benchmarks[1].

2) MNIST: This is a popular dataset for image classification [16]. It contains $70,000$ gray-scale images of size $28x28$, where the original integer pixel values ($[0, 255]$) are re-scaled to the floating-point range $[0, 1]$. We have evaluated CEG4N on nine NNs, from which three models contain a single layer with 10, 25, and 50 ReLU nodes, following the architecture described by Eleftheriadis *et al.* [13]. Three other models, obtained from the VNN-COMP 2021's benchmarks, have 2, 4, and 6 layers, each with 256 ReLU nodes. The remaining three models were trained using resized MNIST images, similar to the first three single-layer ones we described here. In addition, we have employed $8x8$ resized images to reduce dimensionality and also give invariance to small image distortions. In summary, three of the mentioned models were pre-trained, the VNN-COMP 2021's ones, while the remaining elements were trained specifically for our experiments.

3) Seeds: This dataset consists of 210 samples of wheat grain belonging to three different species, namely Kama, Rosa, and Canadian [19]. Its input features include seven measurements of the wheat kernel geometry scaled between $[0, 1]$. We have evaluated CEG4N on four NNs containing a single layer with 4, 6, 10, and 15 ReLU nodes. These four NNs were specifically trained for evaluating CEG4N.

4) Iris: This dataset consists of 50 samples from three species of Iris flower (*Iris setosa, Iris virginica, and Iris versicolor*) [18]. It is a popular benchmark in machine learning for classification, where data is composed of records of real value measurements of the width and length of sepals and petals of flowers. Its data were scaled to $[0, 1]$. We have evaluated CEG4N on three NNs containing two layers with 4, 10, and 15 ReLU nodes in each. These three NNs were trained specifically for evaluating CEG4N.

5) CIFAR-10: This is a dataset for image classification composed by $60,000$ color images with $32x32$ pixels for 10 different classes [17]. We have evaluated CEG4N on two pre-trained NNs made fromVNN-COMP 2021. One has 3 convolutional and 2 linear layers, each with 250 neurons, while the other has 2 convolutional and 2 linear layers, each with 250 neurons. Both use only ReLU activations.

B. Setup

1) BSM: As explained in Section IV-A, we use a search-based optimization algorithm to find bit widths for NN quantization. We have experimented with non-dominated sorting genetic algorithm II (NSGA-II) [47], with the lower and upper bounds for the allowed bit widths set to 2 and 32. The choice for the lower bound relies on the first valid integer that does not break our quantization scheme. In addition, the mentioned upper bound was selected to match the maximum number of bits usually employed for integer representation in many different NN frameworks, such as PyTorch and ONNX, and programming languages, such as C. Moreover, the upper bound could also be higher, depending on the precision of NN weights. However, the single-precision floating-point format is the standard choice to train NNs and store their weights. Another important factor backing our choice is that some NN frameworks may not support other floating-point precision types, which is the case of ONNX. Since the NNs used in our experiments are stored using it, their associated floating-point values are actually represented with single precision.

Furthermore, we have allowed our GA instance to run for 1000 generations for every time BSM was run. Such a figure was found by empirically and incrementally checking if it was able to produce QNNs in initial experiments with our benchmarks. That was done by providing an empty $\mathcal{H}_{\text{CE}}$ to BSM and a given number of generations: if the outputted $\mathcal{N}$ matches a vector such that $n = 2\ \forall n \in \mathcal{N}$, then GA can find a solution with the given number of generations. In our case, GA was able to find solutions for every benchmark when allowing

[1]The pre-trained weight for the ACAS Xu benchmarks can be found in the following repository: https://github.com/stanleybak/vnncomp2021

it to run for 1000. Notice that this number of generations may not be optimal for smaller NNs, but it does not negatively impact the correctness of CEG4N.

Lastly, we randomly selected the initial set $\mathcal{H}^0_{\text{CE}}$ for each dataset, with one sample for each class (e.g., we have selected 10 samples for MNIST and 3 for Iris). The samples in $\mathcal{H}^0_{\text{CE}}$ do not necessarily have to be selected from the benchmarks dataset (train or test), and any concrete input can indeed be specified (e.g., synthetic data). Specifically, in our experiments, we have employed only real data, i.e., data from the chosen datasets, Consequently, we have opted to keep one sample per class. In summary, our choice is further justified by three conditions: (1) the practical aspect of using samples from the benchmarks set, (2) a test set that holds representative data, and (3) the attempt to add variability to our experiments.

2) Equivalence properties: Regarding equivalence, we need to choose from input samples and input constraints of each property, under Definitions 2.1 and 2.2.

The equivalence properties (EPs) for Iris, Seeds, MNIST, and CIFAR-10 were defined by: (1) selecting one real input sample for each class (similar to the definition of $\mathcal{H}^0_{\text{CE}}$), at random; (2) choosing *Top-Equivalence*; and (3) setting input constraints. Regarding the latter, we have defined three possible values for r (see Eq. 13): $r = [0.01, 0.03, 0.05]$. Such values were selected empirically, based on input data and experiments, and reflect the full structure here: benchmarks with narrow input range and *Top-Equivalence*, which usually leads to tighter input regions. In addition, studies in the literature usually adopt only one, while our work provides a margin for its discussion. Taking as an example MNIST, which has 10 output classes, we were able to define a total of 3 sets with 10 input constraints each.

For Acas Xu, we followed the same strategy used by Teuber *et al.* [12], i.e., ϵ-*Equivalence* as equivalence form, while $r = [0.1, 0.3, 0.5]$. Again, three different values were adopted for r, also empirically, but now taking into account Acas Xu's aspects: broader input range and ϵ-*Equivalence*.

If we revisit the Definition 2.2, we must choose two additional parameters, namely p and ϵ. The value for ϵ is usually chosen according to the application domain of the NNs being verified, in such a way that it is possible to prove equivalence and, at the same time, the resulting NNs are useful, i.e., they present tolerable output differences. It is also possible to find an optimal ϵ by incrementally looking at counterexamples and deciding if, from the user perspective, their outputs are equivalent[46]. Ultimately, we decided for $\epsilon = 0.05$ as it means a maximum difference of 10% in Acas Xu's scores. In addition, such a value was also adopted by Teuber *et al.* [12]. Finally, we chose $p = \infty$ due to efficiency reasons [48] and also aimed at consistency across different verifiers, which was also adopted by Teuber *et al.* [12].

3) Time Limits: A timeout is important to ensure termination and should not be arbitrary. In our case, each equivalence property verification takes at most 20 minutes, which is consistent across all verifiers used in our experiments. It was based on hardware configuration, expected run time, and other aspects, but different limits can be set to suit distinct scenarios.

4) Availability of Data and Tools: Our experiments are based on publicly available benchmarks. All tools, benchmarks, and results employed here are available on the supplementary web page https://zenodo.org/record/7126601.

C. Objectives

This work explores the concept of EQ, i.e., a safety property that defines an equivalence relation between an original NN and its quantized form. Then, we propose CEG4N, a framework that quantizes NNs while accounting for EQ.

EG1 Is the CEG4N framework able to generate QNNs that respect the EQ concept?
EG2 How is CEG4N comparable to other quantization techniques?

Due to our research's novelty, no existing similar techniques lend themselves to a fair comparison. Indeed, the present framework is a pioneer one and intends to pave the way for integrating compression into practical NN deployment cycles.

D. Results

In our first set of experiments, we want to achieve our first experimental goal **EG1**. We want to show that CEG4N can successfully generate QNNs that are verifiably equivalent to their original counterparts, i.e., respecting the EQ concept. As secondary goals, we want to: 1) perform an empirical scalability study to help us evaluate the computational demands for quantizing and verifying the equivalence of NN models, and (2) evaluate different equivalence checking techniques and their impacts on the performance of CEG4N.

Our findings are summarized in Tables I, II, III, IV, and V, which show results for benchmarks Iris, Seeds, MNIST, CIFAR, and ACAS Xu, respectively. Regarding the columns available in each Table, *Model* tells the specific benchmark, *Verifier* informs if ESBMC or NNEQUIV was used, r discloses the parameter r (see Eq. (13)), *No. Iter.* shows the number of iterations taken by our GA instance, *Bits* informs the last bit width passed to VM, and *Status* tells the final CEG4N's status. Moreover, the proposed methodology presents inherent flexibility, such that verifiers and constraints can be easily changed and promptly evaluated, as follows.

Our experiments show results for CEG4N executions classified into four possible outcomes: 1) *Success (S)*, meaning that CEG4N ran for one or more iterations and was able to produce a QNN that respects the EQ property; 2) *Timeout (TO)*, which means CEG4N was unable to verify the equivalence property within a given time limit previously set; 3) *Quantization Failure (QF)*, which means that CEG4N was unable to find a suitable set of bit widths to quantize a given NN; and 4) *Verification Failure (VF)*, which means that some error occurred during the equivalence verification step, e.g., exceptions thrown by the VM have occurred.

In summary, CEG4N running with ESBMC (CEG4N+ESBMC) was able to successfully generate QNNs for 17 out of 81 runs, considering all datasets,

TABLE I
SUMMARY OF THE CEG4N'S RESULTS FOR THE IRIS BENCHMARK.

Model	Verifier	r	No. Iter.	Bits	Status
iris_4x2	ESBMC	0.01	1	3,3,3	S
		0.03	1	3,3,3	S
		0.05	11	12,10,10	S
	NNEQUIV	0.01	1	3,3,3	S
		0.03	1	3,3,3	VF
		0.05	3	4,2,3	S
iris_10x2	ESBMC	0.01	2	3,3,2	S
		0.03	3	4,2,4	S
		0.05	7	8,7,9	S
	NNEQUIV	0.01	2	2,3,4	S
		0.03	3	2,3,4	S
		0.05	2	3,2,4	S
iris_15x2	ESBMC	0.01	1	2,2,3	S
		0.03	1	2,2,3	S
		0.05	8		QF
	NNEQUIV	0.01	1	2,2,3	S
		0.03	1	2,2,3	S
		0.05	1	2,2,3	S

TABLE II
SUMMARY OF THE CEG4N'S RESULTS FOR THE SEEDS BENCHMARK.

Model	Verifier	r	No. Iter.	Bits	Status
seeds_4x1	ESBMC	0.01	1	3,3	S
seeds_4x1	ESBMC	0.01	1	3,3	S
seeds_4x1	ESBMC	0.01	1	3,3	S
		0.03	1	3,3	S
		0.05	9	12,13	S
	NNEQUIV	0.01	1	3,3	S
		0.03	1	3,3	S
		0.05	2	4,4	S
seeds_6x1	ESBMC	0.01	1	4,2	S
		0.03	4	12,12	S
		0.05	8	12,12	S
	NNEQUIV	0.01	1	4,2	S
		0.03	2	4,3	S
		0.05	2	4,3	S
seeds_10x1	ESBMC	0.01	2	3,4	S
		0.03	13	18,12	S
		0.05	3	6,4	TO
	NNEQUIV	0.01	2	3,4	S
		0.03	2	4,3	S
		0.05	2	4,4	S
seeds_15x1	ESBMC	0.01	4	5,2	S
		0.03	5	8,6	TO
		0.05	7	8,7	TO
	NNEQUIV	0.01	2	5,2	S
		0.03	2	4,4	S
		0.05	3	5,3	S

TABLE III
SUMMARY OF THE CEG4N'S RESULTS FOR THE MNIST BENCHMARK.

Model	Verifier	r	No. Iter.	Bits	Status
mnist_64_10x1	ESBMC	0.01	1	4,5	TO
		0.03	1	4,5	TO
		0.05	1	4,5	TO
	NNEQUIV	0.01	2	4,5	S
		0.03	2	4,5	S
		0.05	3		QF
mnist_64_25x1	ESBMC	0.01	1	3,3	TO
		0.03	1	3,3	TO
		0.05	1	3,3	TO
	NNEQUIV	0.01	2	5,6	S
		0.03	6	6,5	S
		0.05	4		QF
mnist_64_50x1	ESBMC	0.01	1	2,3	TO
		0.03	1	2,3	TO
		0.05	1	2,3	TO
	NNEQUIV	0.01	2	2,6	S
		0.03	3	5,8	TO
		0.05	1	2,3	TO
mnist_784_10x1 F	ESBMC	0.01	1	3,4	TO
		0.03	1	3,4	TO
		0.05	1	3,4	TO
	NNEQUIV	0.01	1	3,4	S
		0.03	2	3,6	S
		0.05	7	17,21	S
mnist_784_25x1	ESBMC	0.01	1	3,5	TO
		0.03	1	3,5	TO
		0.05	1	3,5	TO
	NNEQUIV	0.01	1	3,5	S
		0.03	1	3,5	TO
		0.05	1	3,5	TO
mnist_784_50x1	ESBMC	0.01	1	3,2	TO
		0.03	1	3,2	TO
		0.05	1	3,2	TO
	NNEQUIV	0.01	1	3,2	S
		0.03	1	3,2	TO
		0.05	1	3,2	TO
mnist_256x2	ESBMC	0.01	1	4,3,5	TO
		0.03	1	4,3,5	TO
		0.05	1	4,3,5	TO
	NNEQUIV	0.01	1	4,3,5	TO
		0.03	1	4,3,5	TO
		0.05	1	4,3,7	TO
mnist_256x4	ESBMC	0.01	1	3,2,3,3,4	TO
		0.03	1	3,2,3,3,4	TO
		0.05	1	3,2,3,3,4	TO
	NNEQUIV	0.01	1	3,2,3,3,4	TO
		0.03	1	3,2,3,3,4	TO
		0.05	1	3,2,3,3,4	TO
mnist_256x6	ESBMC	0.01	1	3,2,3,3,4,3,4	TO
		0.03	1	3,2,3,3,4,3,4	TO
		0.05	1	3,2,3,3,4,3,4	TO
	NNEQUIV	0.01	1	3,2,3,3,4,3,4	TO
		0.03	1	3,2,3,3,4,3,4	TO
		0.05	1	3,2,3,3,4,3,4	TO

which accounts for 20.99% of all processes. In addition, CEG4N running with NNEQUIV (CEG4N+NNEQUIV) was successful in 33 out of 81 runs, representing 40.74% of all processes. Most of the CEG4N's failures, with ESBMC, was due to timeouts, with 55 occurrences, representing 67.90% of the total. In contrast, CEG4N with NNEQUIV resulted in 30 timeouts, i.e., 37% of the total.

Such a difference in timeouts can be attributed to many factors. For example, ESBMC, as an approach based on bounded model checking (BMC), is known to suffer from scalability issues, which greatly diminishes its ability to support larger NNs [29] and can be seen in more details in Tables I, II, III, IV, and V. In the first two, which show information regarding CEG4N+ESBMC runs for the Iris and Seeds datasets (at most two layers with less than 20 node each), respectively, only three timeouts were noticed. However, if we take a look at Tables III, IV, and V, which show information regarding CEG4N+ESBMC runs for MNIST, CIFAR-10, and AcasXu, respectively, and are composed mostly by medium to large NNs (at most eight layers), the number of timeouts presents a significant increase. Indeed, no successful execution could even be identified. Moreover, the rare runs with a timeout, with the dataset Seeds processed by CEG4N+ESBMC, also happened with (1) NNs containing more than 10 neurons per layer, (2) more than 4 iterations, and (3) larger constraint regions ($r = 0.03$ and 0.05), which reinforces the explanation

TABLE IV
SUMMARY OF THE CEG4N'S RESULTS FOR THE CIFAR BENCHMARK.

Model	Verifier	r	No. Iter.	Bits	Status
cifar10_2_255	ESBMC	0.01	1	9,5,6,6,6	TO
		0.03	1	7,8,6,6,6	TO
		0.05	1	10,5,6,6,6	TO
	NNEQUIV	0.01	1	7,8,6,6,6	VF
		0.03	1		QF
		0.05	1	9,5,6,7,7	VF
cifar10_8_255	ESBMC	0.01	1	4,8,6,7	TO
		0.03	1	6,8,6,8	TO
		0.05	1	4,8,6,7	TO
	NNEQUIV	0.01	1	4,8,6,7	VF
		0.03	1	4,8,6,7	VF
		0.05	1		QF

based on NN complexity and BMC scalability. We can check such cases in detail in Table II. In summary, combining factors (1) and (3) mostly contributes to timeouts. The more neurons an NN has, the more operations it has to perform (and CEG4N). Besides, the bigger the r value, the bigger the search space a verifier has to cover.

Although runs with CEG4N+NNEQUIV suffered from fewer timeouts, it's interesting to notice that CEG4N+ESBMC required overall more iterations than CEG4N+NNEQUIV. Considering only the successful runs, CEG4N+ESBMC needed, on average, 4 iterations to produce a QNN, while CEG4N+NNEQUIV used 2. The explanation for this behavior is that ESBMC can find counterexamples that NNEQUIV is not. Since ESBMC and NNEQUIV use different verification approaches, i.e., SMT and reachability analysis, respectively, the obtained results are expected to eventually diverge for some verification instances. In addition, NNEQUIV produces a high number of spurious counterexamples, which is not beneficial to our scheme. Indeed, CEG4N already expects that the verifiers can eventually produce spurious counterexamples, which are ruled out as explained in Section IV-B3. Therefore, our results are entirely based on valid counterexamples only.

Quantization failures are another possible type of error, which can occur only in BSM. One may notice that CEG4N+ESBMC presented only 9 cases, i.e., 11.11%, while 12 were identified for CEG4N+NNEQUIV. i.e., 14.81%. A possible explanation is that the search for the bits sequence is highly non-linear and highly dependent on the set of counterexamples $\mathcal{H}_{CE}$ BSM receives at a given iteration, which makes this optimization problem a hard one to solve. We should also consider that new constraints are added to the search problem after each iteration, which makes it even harder to solve. This is corroborated by the fact that most quantization failures occurred after 2 or more runs.

Our experiments show a high number of quantization-failure events for Acas Xu's benchmarks. One explanation for that relies on the fact that some of those NNs are highly sensitive to errors introduced by quantization processes, in such a way that it affects those NN's behaviors. Thus, in such a context, NNs may easily violate the constraint $f(x) \simeq f^q(x), \ \forall\ x \in \mathcal{H}^o_{\text{CE}}$ during the step performed by BSM. In addition, for other NNs,

TABLE V
SUMMARY OF THE CEG4N'S RESULTS FOR THE ACAS XU BENCHMARK.

Model	Verifier	r	No. Iter.	Bits	Status
ACASXU_1_1	ESBMC	0.1	1		QF
		0.3	1		QF
		0.5	1		QF
	NNEQUIV	0.1	1		QF
		0.3	1		QF
		0.5	1	10,9,9,7,12,10,8	S
ACASXU_1_2	ESBMC	0.1	1	12,9,8,9,7,9,5	TO
		0.3	1		QF
		0.5	1	11,7,7,8,9,9,7	TO
	NNEQUIV	0.1	1		QF
		0.3	1	7,8,7,8,9,9,5	TO
		0.5	1		QF
ACASXU_1_3	ESBMC	0.1	1		QF
		0.3	1		QF
		0.5	1		QF
	NNEQUIV	0.1	1		QF
		0.3	1		QF
		0.5	1	8,7,8,8,8,9,6	TO
ACASXU_1_4	ESBMC	0.1	1	5,6,5,6,6,8,4	TO
		0.3	1	6,6,7,6,6,7,4	TO
		0.5	1		QF
	NNEQUIV	0.1	1		QF
		0.3	1	5,6,5,6,6,8,4	TO
		0.5	1	7,6,6,6,6,7,4	S
ACASXU_1_5	ESBMC	0.1	1	5,6,6,6,6,7,4	TO
		0.3	1	5,6,6,6,6,7,4	TO
		0.5	1	8,6,7,5,5,7,4	TO
	NNEQUIV	0.1	1	7,8,5,6,7,7,8	TO
		0.3	1	5,6,6,6,6,7,4	S
		0.5	1		QF
ACASXU_1_6	ESBMC	0.1	1	2,2,2,2,2,2,2	TO
		0.3	1	5,9,4,2,6,4,4	TO
		0.5	1	2,2,2,2,2,6,4	TO
	NNEQUIV	0.1	1	2,2,2,2,2,2,2	TO
		0.3	1	2,2,2,2,2,2,2	TO
		0.5	1	2,2,2,2,2,6,4	TO
ACASXU_1_7	ESBMC	0.1	1	2,2,2,2,2,4,2	TO
		0.3	1	2,2,2,2,2,2,2	TO
		0.5	1	2,2,2,2,2,2,2	TO
	NNEQUIV	0.1	1	2,2,2,2,2,2,2	TO
		0.3	1	2,2,2,2,2,2,2	VF
		0.5	1	2,2,2,2,2,2,2	TO
ACASXU_1_8	ESBMC	0.1	1	2,2,2,2,2,2,2	TO
		0.3	1	2,2,2,2,2,2,2	TO
		0.5	1	2,2,2,2,2,2,2	TO
	NNEQUIV	0.1	1	2,2,2,2,2,2,2	TO
		0.3	1	2,2,2,2,2,2,2	TO
		0.5	1	2,2,2,2,2,2,2	TO
ACASXU_1_9	ESBMC	0.1	1	2,2,2,2,2,2,2	TO
		0.3	1	2,2,2,2,2,2,2	TO
		0.5	1	2,2,2,2,2,2,2	TO
	NNEQUIV	0.1	1	2,2,2,2,2,2,2	TO
		0.3	1	2,2,2,2,2,2,2	TO
		0.5	1	2,2,2,2,2,2,2	TO

a higher value for ϵ can also help increase the chance of a successful quantization. However, in the specific scenario used here, we used an ϵ that should result in applicable QNNs [12].

Indeed, the conditions presented in the last paragraph probably prevented CEG4N from producing QNN candidates for VM. Moreover, successful runs mostly failed with timeouts. The main factors behind the latter are (1) the size of the input NNs, in neurons, and (2) the number of features in the input space. Indeed, Acas Xu's NNs are mostly affected by the NN size problem, as they present 300 neurons in total. MNIST, in turn, NNs have 64 or 784 features in the input space that, from the verification perspective, resulting in a very large search space dimension to cover. In addition, CIFAR-10's NNs are affected by both factors, having 1024 features in the input space, and thousands of neurons in total.

Regarding ACAS Xu, we find it possible to tune p and ϵ so we mitigate most of the associated quantization failures. However, it should be done carefully so the resulting NNs are still applicable and the chosen verifier can perform accordingly.

The last possible failures are the verification ones, which only occur in VM. We have noticed them only when running CEG4N+NNEQUIV, with a total of 6 cases, i.e., a failure rate of 7.4%. They were caused by exceptions thrown by the NNEQUIV's software dependencies. However, it is unclear that a flaw in NNEQUIVcaused those exceptions. As we did not notice any verification failures with ESBMC, this can indicate its maturity, when comparing it with NNEQUIV.

Finally, the VM choice is also important. As shown here, CEG4N+ESBMC and CEG4N+NNEQUIV present different behaviors, which may be suitable for a given scenario. For instance, regarding large NNs, CEG4N+NNEQUIV seems to be a clear choice. Moreover, if new verification techniques are introduced, one may consider other VM options, which our framework can accommodate due to its modularity.

These results answer our **EG1**. Overall, these experiments show that CEG4N can successfully produce equivalent QNNs. However, scalability should be a point of concern for larger NN models, which is very related to the chosen verifier.

In our second set of experiments, we want to achieve our second experimental goal, i.e., **EG2**. We primarily want to understand the impact of quantization processes performed by CEG4N on the accuracy of the resulting NNs compared with other post-training quantization techniques.

We selected the models for which CEG4N presented successful quantization processes and proceeded to quantize them also using GPFQ. Next, we collected the accuracy of the original and QNNs to compare them. Tables VI VII, and VIII summarize the accuracy of the models quantized with CEG4N+NNEQUIV, CEG4N+ESBMC, and GPFQ, using the Iris, Seeds, and MNIST benchmarks. However, there was no successful run regarding CEG4N+ESBMC with MNIST, as already mentioned. Note that we do not report the accuracy of CIFAR-10's NNs from VNNCOMP since CEG4N could not quantize them. Also, we do not report the accuracy of Acas Xu's models. It happened partially because of the same problem identified with the VNNCOMP's models, but mainly because GPFQ requires access to training and test datasets, which are not public for Acas Xu.

The tables are organized as follows. Column *Model* shows the name of the NN models, *Quantizer* tells the name quantization technique (i.e., CEG4N+ESBMC, CEG4N+NNEQUIV, or GPFQ), r informs the value used to define quantization EPs, and columns *Original Accuracy*, *Quantized Accuracy*, and *Accuracy Drop* show the accuracy of the original and QNNs and their difference, respectively. Lastly, the column *Equivalence Status* tells whether original and QNNs are equivalent. The *Accuracy Drop* is positive if the QNN has a worse accuracy when compared with its original model. Otherwise, it is negative.

Our findings show that the highest drops in accuracy for NNs generated by CEG4N+nnequiv where 3.3% for Iris, 21.43% for Seeds, and -6.20% for MNIST. For NNs quantized with CEG4Nesbmc, the highest drops were 3.3% for Iris and 21% for Seeds. Regarding GPFQ, the highest noticed drops were 53.3% for Iris, -47.61% for Seeds, and -3.41% for MNIST. Such results are interesting and show that an increase in accuracy is even possible, which will be briefly discussed in the following text.

TABLE VI
COMPARISON USING TOP-1 ACCURACY FOR NNS FROM DATASET IRIS QUANTIZED USING CEG4N AND GPFQ

Model	Quantizer	r	Original Accuracy	Quantized Accuracy	Accuracy Drop	Equivalence Status
iris_4x2	GPFQ	0.010	86.667	60.000	**-26.667**	True
iris_4x2	CEG4N+ESBMC	0.010	86.667	86.667	0.000	True
iris_4x2	CEG4N+NNEQUIV	0.010	86.667	86.667	0.000	True
iris_4x2	GPFQ	0.030	86.667	60.000	**-26.667**	True
iris_4x2	CEG4N+ESBMC	0.030	86.667	86.667	0.000	True
iris_4x2	GPFQ	0.050	86.667	40.000	**-46.667**	False
iris_4x2	CEG4N+ESBMC	0.050	86.667	86.667	0.000	True
iris_4x2	CEG4N+NNEQUIV	0.050	86.667	93.333	6.667	True
iris_10x2	GPFQ	0.010	90.000	36.667	**-53.333**	False
iris_10x2	CEG4N+ESBMC	0.010	90.000	96.667	6.667	True
iris_10x2	CEG4N+NNEQUIV	0.010	90.000	93.333	3.333	True
iris_10x2	GPFQ	0.030	90.000	36.667	**-53.333**	False
iris_10x2	CEG4N+ESBMC	0.030	90.000	86.667	-3.333	True
iris_10x2	CEG4N+NNEQUIV	0.030	90.000	93.333	3.333	True
iris_10x2	GPFQ	0.050	90.000	36.667	**-53.333**	True
iris_10x2	CEG4N+ESBMC	0.050	90.000	90.000	0.000	True
iris_10x2	CEG4N+NNEQUIV	0.050	90.000	96.667	6.667	True
iris_15x2	GPFQ	0.010	90.000	93.333	3.333	False
iris_15x2	CEG4N+ESBMC	0.010	90.000	86.667	-3.333	True
iris_15x2	CEG4N+NNEQUIV	0.010	90.000	86.667	-3.333	True
iris_15x2	GPFQ	0.030	90.000	93.333	3.333	False
iris_15x2	CEG4N+ESBMC	0.030	90.000	86.667	-3.333	True
iris_15x2	CEG4N+NNEQUIV	0.030	90.000	86.667	-3.333	True
iris_15x2	GPFQ	0.050	90.000	93.333	3.333	False
iris_15x2	CEG4N+NNEQUIV	0.050	90.000	86.667	-3.333	True

TABLE VII
COMPARISON USING TOP-1 ACCURACY FOR NNS FROM DATASET SEEDS QUANTIZED USING CEG4N AND GPFQ

Model	Quantizer	r	Original Accuracy	Quantized Accuracy	Accuracy Drop	Equivalence Status
seeds_4x1	GPFQ	0.010	88.095	64.286	**-23.810**	True
seeds_4x1	CEG4N+ESBMC	0.010	88.095	85.714	-2.381	True
seeds_4x1	CEG4N+NNEQUIV	0.010	88.095	85.714	-2.381	True
seeds_4x1	GPFQ	0.030	88.095	64.286	**-23.810**	True
seeds_4x1	CEG4N+ESBMC	0.030	88.095	85.714	-2.381	True
seeds_4x1	CEG4N+NNEQUIV	0.030	88.095	85.714	-2.381	True
seeds_4x1	GPFQ	0.050	88.095	64.286	**-23.810**	True
seeds_4x1	CEG4N+ESBMC	0.050	88.095	88.095	0.000	True
seeds_4x1	CEG4N+NNEQUIV	0.050	88.095	88.095	0.000	True
seeds_6x1	GPFQ	0.010	83.333	59.524	**-23.810**	False
seeds_6x1	CEG4N+ESBMC	0.010	83.333	73.810	-9.524	True
seeds_6x1	CEG4N+NNEQUIV	0.010	83.333	73.810	-9.524	True
seeds_6x1	GPFQ	0.030	83.333	85.714	2.381	False
seeds_6x1	CEG4N+ESBMC	0.030	83.333	83.333	0.000	True
seeds_6x1	CEG4N+NNEQUIV	0.030	83.333	80.952	-2.381	True
seeds_6x1	GPFQ	0.050	83.333	57.143	**-26.190**	False
seeds_6x1	CEG4N+ESBMC	0.050	83.333	83.333	0.000	True
seeds_6x1	CEG4N+NNEQUIV	0.050	83.333	80.952	-2.381	True
seeds_10x1	GPFQ	0.010	90.476	42.857	**-47.619**	False
seeds_10x1	CEG4N+ESBMC	0.010	90.476	90.476	0.000	True
seeds_10x1	CEG4N+NNEQUIV	0.010	90.476	90.476	0.000	True
seeds_10x1	GPFQ	0.030	90.476	80.952	-9.524	False
seeds_10x1	CEG4N+ESBMC	0.030	90.476	90.476	0.000	True
seeds_10x1	CEG4N+NNEQUIV	0.030	90.476	90.476	0.000	True
seeds_10x1	GPFQ	0.050	90.476	80.952	-9.524	True
seeds_10x1	CEG4N+NNEQUIV	0.050	90.476	90.476	0.000	True
seeds_15x1	GPFQ	0.010	90.476	76.190	**-14.286**	False
seeds_15x1	CEG4N+ESBMC	0.010	90.476	69.048	**-21.429**	True
seeds_15x1	CEG4N+NNEQUIV	0.010	90.476	69.048	**-21.429**	True
seeds_15x1	GPFQ	0.030	90.476	88.095	-2.381	True
seeds_15x1	CEG4N+NNEQUIV	0.030	90.476	90.476	0.000	True
seeds_15x1	GPFQ	0.050	90.476	78.571	-11.905	False
seeds_15x1	CEG4N+NNEQUIV	0.050	90.476	85.714	-4.762	True

TABLE VIII
COMPARISON USING TOP-1 ACCURACY FOR NNS FROM DATASET MNIST QUANTIZED USING CEG4N AND GPFQ

Model	Quantizer	r	Original Accuracy	Quantized Accuracy	Accuracy Drop	Equivalence Status
mnist_64_10x1	GPFQ	0.010	79.040	79.480	0.440	False
mnist_64_10x1	CEG4N+NNEQUIV	0.010	79.040	77.900	-1.140	True
mnist_64_10x1	GPFQ	0.030	79.040	79.480	0.440	False
mnist_64_10x1	CEG4N+NNEQUIV	0.030	79.040	77.900	-1.140	True
mnist_64_25x1	GPFQ	0.010	83.420	81.960	-1.460	False
mnist_64_25x1	CEG4N+NNEQUIV	0.010	83.420	82.820	-0.600	True
mnist_64_25x1	GPFQ	0.030	83.420	81.960	-1.460	False
mnist_64_25x1	CEG4N+NNEQUIV	0.030	83.420	83.220	-0.200	True
mnist_64_50x1	GPFQ	0.010	84.600	82.410	-2.190	False
mnist_64_50x1	CEG4N+NNEQUIV	0.010	84.600	77.280	-7.320	True
mnist_784_10x1	GPFQ	0.010	90.260	86.850	-3.410	False
mnist_784_10x1	CEG4N+NNEQUIV	0.010	90.260	89.170	-1.090	True
mnist_784_10x1	GPFQ	0.030	90.260	86.850	-3.410	False
mnist_784_10x1	CEG4N+NNEQUIV	0.030	90.260	89.940	-0.320	True
mnist_784_10x1	GPFQ	0.050	90.260	90.330	0.070	True
mnist_784_10x1	CEG4N+NNEQUIV	0.050	90.260	90.260	0.000	True
mnist_784_25x1	GPFQ	0.010	91.940	91.660	-0.280	False
mnist_784_25x1	CEG4N+NNEQUIV	0.010	91.940	91.350	-0.590	True
mnist_784_50x1	GPFQ	0.010	92.150	91.240	-0.910	False
mnist_784_50x1	CEG4N+NNEQUIV	0.010	92.150	85.950	-6.200	True

On the one hand, the highest accuracy drop for CEG4N-generated NNs coincides with $r = 0.01$ (See Table VI). Indeed, small r values increase the chance of not finding counterexamples to drive the quantization process, which favors the generation of poorly QNNs. For higher r values ($r = 0.03$ and $r = 0.05$), we notice that the highest accuracy drop is 3.3%. On the other hand, for GPFQ, the highest accuracy drops involve the Iris and Seeds benchmarks. We believe the GPFQ's performance is due to two factors: 1) GPFQ relies on representative data (e.g., data from the training dataset), which is more difficult for small datasets; and 2) small models are more sensitive to quantization.

Overall, the accuracy of models quantized with CEG4N is better for the Iris and Seeds benchmarks, while the accuracy of models quantized with GPFQ is better for the MNIST benchmarks, but only by a small margin. We find that the CEG4N's performance, in terms of NN accuracy, presents interesting results, given that it can produce QNNs relying only on a small set of representative data.

We have also conducted another experiment in which NNEQUIV was used to verify equivalence between QNNs generated by GPFQ and their original counterparts. For every case, when at least one counterexample was found, we considered the QNN not equivalent to the original one. We have noticed that, out of 31 NNs generated by GPFQ, only 8 were in fact equivalent, which represents only 25.8% of the total amount of resulting NNs. Indeed, GPFQ was not designed to consider EPs in its quantization approach as happened with CEG4N. Anyway, these experiments serve as evidence that statistical accuracy measures do not capture equivalence aspects. Moreover, it reinforces the benefits of formulating guarantees for properties (e.g, equivalence and robustness) that formal verification techniques can offer.In addition, we noticed that both CEG4N and GPFQ produced QNNs with better accuracy when compared with their original counterparts. Indeed, that is possible and has already been reported in the literature, since the quantization techniques act as favorable weight regularization mechanisms that help NNs prevent biased behavior and provide better generalization [49].

These results answer our **EG2**. Overall, these experiments show that CEG4N can successfully produce QNNs that present accuracy figures similar to what is obtained with other state-of-the-art techniques.

E. Limitations

Although CEG4N can generate QNNs while keeping equivalence with the original ones, the NNs we have used for evaluation may not fully reflect the state-of-the-art. Indeed, they have a few layers and hundreds of ReLU nodes, while state-of-the-art ones may have hundreds of layers and thousands of ReLU nodes. The main bottleneck lies in the state-of-the-art verification algorithms (e.g., SMT or GPE), which currently do not scale to large NNs. Consequently, CEG4N+ESBMC and CEG4N+NNEQUIV were able to quantize only 20% and 40% of the chosen benchmarks, respectively, due to timeouts. Still, we have shown that CEG4N is both viable and flexible, which opens room for improvements with verifiers beyond ESBMC and NNEQUIV.

The research field of NN equivalence is a relatively new one and there is no well-established set of benchmarks [13]. In this respect, our choices are a good starting point, but there is ample scope for further contributions. Additionally, our work is innovative by proposing a framework for NN quantization that integrates NN equivalence verification as an essential part of the process. As such, there is no similar methodology in literature which we can directly compare our approach with. Also, our work focuses on the practical and feasible aspect of this new quantization approach and the formalization of NN equivalence in terms of functional equivalence. Thus, we are not centering the discussion at a conceptual level, and for such, we could not discuss in depth the relationship between NN equivalence, functional equivalence, robustness verification, their impact, and implication in NN accuracy and error.

VI. CONCLUSION

We have presented a new method for NN quantization, named CEG4N, which is a post-training quantization technique that provides formal guarantees regarding NN equivalence. It relies on a counterexample-guided optimization technique, where an optimization-based quantizer produces compressed NN candidates. A state-of-the-art verifier then checks such candidates to either prove the equivalence between quantized and original NNs or refute it by providing a counterexample. The latter is then fed to the quantizer to guide it in the search for a viable candidate.

In the proposed framework, scalability is tightly related to the underlying verifier, which, in our experiments, took two forms: SMT solver and GPE. SMT solvers are sensitive to NN complexity [29], which leads to large state spaces, while GPE may face exponential growth in the number of associated star sets [12]. Although that may look like an obstacle, both optimization efforts and different quantization strategies have the potential to alleviate these issues. At the same time, it is worth mentioning that our main target was to demonstrate the feasibility of our methodology, which employed the mentioned verifiers as possible solutions, but it is not restricted to them.

In our future work, we will explore other quantization approaches not limited to the search-based ones and different equivalence-verification techniques based on reachability analysis [12] and SMT encoding [13]. For instance [39] provides a new perspective and interpretation of NN behavior centered around error bound verification. This interpretation is different than ours and may have implications and provide new perspectives and conclusions to our quantization problem. Possible future works may provide a more concise formalization of NN equivalence, incorporate QEBVerif approach into CEG4N and provide a comparison work. Combining new quantization and equivalence-verification techniques will help CEG4N achieve better results while providing a more suitable compromise between accuracy and scalability. Furthermore, we will consider quantization approaches that operate entirely on integer arithmetic, which can potentially improve the scalability of the CEG4N's verification step. The SMT-encoding of the quantization problem will also be considered, with the goal of both comparing its cost with the one presented by our current proposal and devising a unified framework for NN compression and equivalence.

References

[1] M. Bojarski, D. W. del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *ArXiv:abs/1604.07316*, 2016.

[2] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018.

[3] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *ArXiv:abs/1710.09282*, 2017.

[4] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, 2015.

[5] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

[6] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," in *5th International Conference on Learning Representations*. OpenReview.net, 2017.

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

[8] J. Zhang, Y. Zhou, and R. Saab, "Post-training quantization for neural networks with provable guarantees," *SIAM Journal on Mathematics of Data Science*, vol. 5, no. 2, pp. 373–399, 2023.

[9] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," *in CVPR*, pp. 2704–2713, 2018.

[10] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Computer Science Review*, vol. 37, p. 100270, 2020.

[11] M. K. Büning, P. Kern, and C. Sinz, "Verifying equivalence properties of neural networks with relu activation functions," in *Principles and Practice of Constraint Programming*, ser. Lecture Notes in Computer Science, vol. 12333. Springer, 2020, pp. 868–884.

[12] S. Teuber, M. K. Buning, P. Kern, and C. Sinz, "Geometric path enumeration for equivalence verification of neural networks," *in ICTAI*, Nov 2021.

[13] C. Eleftheriadis, N. Kekatos, P. Katsaros, and S. Tripakis, "On neural network equivalence checking using SMT solvers," in *Formal Modeling and Analysis of Timed Systems*. Cham: Springer, 2022, pp. 237–257.

[14] J. B. P. Matos, I. Bessa, E. Manino, X. Song, and L. C. Cordeiro, "CEG4N: Counter-example guided neural network quantization refinement," in *Software Verification and Formal Methods for ML-Enabled Autonomous Systems*. Cham: Springer, 2022, pp. 29–45.

[15] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *DASC*, 2016, pp. 1–10.

[16] Y. LeCun and C. Cortes, "The MNIST database of handwritten digits," 2005.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012.

[18] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, pp. 179–188, 1936.

[19] M. Charytanowicz, J. Niewczas, P. Kulczycki, P. A. Kowalski, S. Łukasik, and S. Żak, *Complete Gradient Clustering Algorithm for Features Analysis of X-Ray Images*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–24.

[20] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[21] A. Abate, I. Bessa, D. Cattaruzza, L. C. Cordeiro, C. David, P. Kesseli, and D. Kroening, "Sound and automated synthesis of digital stabilizing controllers for continuous plants," in *HSCC*. ACM, 2017, pp. 197–206.

[22] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *ArXiv:abs/1806.08342*, 2018.

[23] B. Paulsen, J. Wang, J. Wang, and C. Wang, "NEURODIFF: Scalable differential verification of neural networks using fine-grained approximation," *35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 784–796, 2020.

[24] B. Paulsen, J. Wang, and C. Wang, "Reludiff: Differential verification of deep neural networks," in *ISCE*. Association for Computing Machinery, 2020, p. 714–726.

[25] R. Brummayer and A. Biere, "Boolector: An efficient SMT solver for bit-vectors and arrays," in *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 174–177.

[26] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.

[27] H.-D. Tran, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis of deep neural networks," in *FM*, 2019.

[28] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *ArXiv:abs/2106.08295*, 2021.

[29] L. Sena, X. Song, E. H. da S. Alves, I. V. Bessa, E. Manino, and L. C. Cordeiro, "Verifying quantized neural networks using SMT-based model checking," *ArXiv:abs/2106.05997*, 2021.

[30] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016.

[31] M. Á. Carreira-Perpiñán and Y. Idelbayev, "Model compression as constrained optimization, with application to neural nets. part ii: quantization," *ArXiv:abs/1707.04319*, 2017.

[32] Y. Yuan, C. Chen, X. Hu, and S. Peng, "Evoq: Mixed precision quantization of dnns via sensitivity guided evolutionary search," *2020 International Joint Conference on Neural Networks*, pp. 1–8, 2020.

[33] E. Lybrand and R. Saab, "A greedy algorithm for quantizing neural networks," *J. Mach. Learn. Res.*, vol. 22, pp. 156:1–156:38, 2020.

[34] Y. Xu, Y. Wang, A. Zhou, W. Lin, and H. Xiong, "Deep neural network compression with single and multiple level quantization," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[35] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, "Quantization networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[36] Q. Jin, L. Yang, and Z. Liao, "Adabits: Neural network quantization with adaptive bit-widths," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[37] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *ArXiv*, vol. abs/1606.06160, 2016.

[38] K. Huang, B. Li, D. Xiong, H. Jiang, X. Jiang, X. Yan, L. J. M. Claesen, D. Liu, J. Chen, and Z. Liu, "Structured dynamic precision for deep neural networks quantization," *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, pp. 1 – 24, 2022.

[39] Y. Zhang, F. Song, and J. Sun, "QEBVerif: Quantization error bound verification of neural networks," in *Computer Aided Verification*. Cham: Springer Nature Switzerland, 2023, pp. 413–437.

[40] M. Giacobbe, T. A. Henzinger, and M. Lechner, "How many bits does it take to quantize your neural network?" in *Tools and Algorithms for the Construction and Analysis of Systems*. Cham: Springer International Publishing, 2020, pp. 79–97.

[41] T. A. Henzinger, M. Lechner, and D. Žikelić, "Scalable verification of quantized neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, pp. 3787–3795, May 2021.

[42] S. Mistry, I. Saha, and S. Biswas, "An MILP encoding for efficient verification of quantized deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4445–4456, 2022.

[43] X. Song, E. Manino, L. H. Sena, E. H. da S. Alves, E. B. de Lima Filho, I. Bessa, M. Luján, and L. C. Cordeiro, "QNNVerifier: A tool for verifying neural networks using SMT-based model checking," *ArXiv:abs/2111.13110*, 2021.

[44] M. R. Gadelha, F. R. Monteiro, J. Morse, L. C. Cordeiro, B. Fischer, and D. A. Nicole, "Esbmc 5.0: An industrial-strength c model checker," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018, pp. 888–891.

[45] F. R. Monteiro, M. R. Gadelha, and L. C. Cordeiro, "Model checking c++ programs," *Software Testing, Verification and Reliability*, vol. 32, no. 1, p. e1793, 2022.

[46] S. Bak, C. Liu, and T. Johnson, "The second international verification of neural networks competition (vnn-comp 2021): Summary and results," *ArXiv:abs/2109.00498*, 2021.

[47] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

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

[49] Q. Jin, L. Yang, and Z. A. Liao, "Towards efficient training for neural network quantization," *ArXiv:abs/1912.10207*, 2019.

João Batista P. Matos Jr. is a Ph.D. candidate in the Institute of Computation at the Federal University of Amazonas. He is a research student of the Systems and Software Security group and focuses on automatic verification of neural networks. He has a background in recommendation systems, having studied user-engagement metrics for social-networks.

Eddie B. de Lima Filho is the leader of the digital convergence group in the R&D Department of TPV Technology, Brazil, where he conducts research in Consumer Electronics. In addition, he is affiliated with the Post-Graduate Program in Electrical Engineering (PPGEE) at the Federal University of Amazonas, Brazil. Before joining TPV Technology, he worked as a researcher in the Science, Technology, and Innovation Center for the Manaus Industrial Pole of Manaus / NXP semiconductors and Genius Institute of Technology. His work includes software model checking, automated testing, embedded and cyber-physical systems, channel/source coding, and video/image processing.

Iury Bessa received his B.Sc., in 2014, and Master degrees, in 2015, all in Electrical Engineering from Federal University of Amazonas, Brazil. He received his Ph.D. degree in Electrical Engeneering with the D!FCOM laboratory at Federal University of Minas Gerais. During his Ph.D., he was a visiting scholar at the Advanced Control Systems group, Polytechnic University of Catalonia (UPC), Spain, from February to December of 2020. Since 2015, he is an Assistant Professor at the Department of Electricity and is part of the e-Controls research group, Federal University of Amazonas, Brazil. In October 2016, he was a visiting scholar at the Department of Computer Science, University of Oxford, UK. His research interests include control theory, fault-tolerant control, fault detection, diagnosis and prognosis, formal verification and synthesis, learning-based control, cyber-physical systems, and computational intelligence.

Edoardo Manino is a Research Associate in the Department of Computer Science at the University of Manchester. He is part of the Systems and Software Security group and focuses on automated verification of neural networks. His background is in Bayesian machine learning, a topic he got awarded a PhD from the University of Southampton in 2020. Throughout his research career, he published on a wide number of topics, including algorithmic game theory, multi-agent reinforcement learning, network science, crowdsourcing, analog computing and automated software testing.

Xidan Song is a PhD student in the Department of Computer Science at the University of Manchester. He is a research student of the Systems and Software Security group and focuses on the verification and repair of neural networks.

Lucas C. Cordeiro is a Reader in the Department of Computer Science at the University of Manchester (UoM), where he leads the Systems and Software Security (S3) Research Group. Dr. Cordeiro is also the Arm Centre of Excellence Director at UoM. In addition, he is affiliated with the Trusted Digital Systems Cluster at the Centre for Digital Trust and Society, the Formal Methods Group at UoM, and the Post-Graduate Programs in Electrical Engineering (PPGEE) and Informatics (PPGI) at the Federal University of Amazonas, Brazil. Before joining the University of Manchester, he worked as a post-doctoral researcher at the University of Oxford and as a research engineer at Diffblue. In addition, Dr. Cordeiro worked for five years as a software engineer at Siemens / BenQ Mobile and CTPIM / NXP semiconductors. His work focuses on software model checking, automated testing, program synthesis, software security, embedded and cyber-physical systems. He has co-authored more than 150 peer-reviewed publications in the most prestigious venues (e.g., ICSE, CAV, TACAS, FSE, ASE, ISSTA, TSE, TR, TC). He has received various international awards, including the Most Influential Paper at IEEE/ACM ASE'23, the Distinguished Paper Award at ACM ICSE'11, and 39 awards from the international competitions on software verification (SV-COMP) and testing (Test-Comp) 2012-2023. He has a proven track record of securing research funding from EPSRC, Intel, Motorola, Samsung, Nokia Institute of Technology, CNPq, FAPEAM, British Council, and Royal Society.