# Efficient Certified Reasoning for Binarized Neural Networks

**Jiong Yang**
Georgia Institute of Technology, Atlanta, GA, USA

**Yong Kiam Tan**
Institute for Infocomm Research ($I^2R$), A*STAR, Singapore
Nanyang Technological University, Singapore

**Mate Soos**
University of Toronto, Canada

**Magnus O. Myreen**
Chalmers University of Technology, Gothenburg, Sweden
University of Gothenburg, Sweden

**Kuldeep S. Meel**
Georgia Institute of Technology, Atlanta, GA, USA
University of Toronto, Canada

## Abstract

Neural networks have emerged as essential components in safety-critical applications – these use cases demand complex, yet trustworthy computations. Binarized Neural Networks (BNNs) are a type of neural network where each neuron is constrained to a Boolean value; they are particularly well-suited for safety-critical tasks because they retain much of the computational capacities of full-scale (floating-point or quantized) deep neural networks, but remain compatible with satisfiability solvers for qualitative verification and with model counters for quantitative reasoning. However, existing methods for BNN analysis suffer from either limited scalability or susceptibility to soundness errors, which hinders their applicability in real-world scenarios.

In this work, we present a scalable and trustworthy approach for both qualitative and quantitative verification of BNNs. Our approach introduces a native representation of BNN constraints in a custom-designed solver for qualitative reasoning, and in an approximate model counter for quantitative reasoning. We further develop specialized proof generation and checking pipelines with native support for BNN constraint reasoning, ensuring trustworthiness for all of our verification results. Empirical evaluations on a BNN robustness verification benchmark suite demonstrate that our certified solving approach achieves a 9× speedup over prior certified CNF and PB-based approaches, and our certified counting approach achieves a 218× speedup over the existing CNF-based baseline. In terms of coverage, our pipeline produces fully certified results for 99% and 86% of the qualitative and quantitative reasoning queries on BNNs, respectively. This is in sharp contrast to the best existing baselines which can fully certify only 62% and 4% of the queries, respectively.

**2012 ACM Subject Classification** Theory of computation → Logic and verification; Computing methodologies → Neural networks; Computing methodologies → Artificial intelligence

**Keywords and phrases** Neural network verification, proof certification, SAT solving, approximate model counting

**Digital Object Identifier** 10.4230/LIPIcs.SAT.2025.32

**Related Version** *arXiv Version:*: `https://arxiv.org/abs/2507.02916`

**Supplementary Material**
*Software (CryptoMiniSat-BNN)*: `https://github.com/msoos/cryptominisat`
*Software (FRAT-xor-bnn and cake_xlrup-BNN)*: `https://github.com/meelgroup/frat-xor`
*Software (ApproxMC-BNN)*: `https://github.com/meelgroup/approxmc`
*Software (ApproxMCCert-BNN and CertCheck-BNN)*: `https://github.com/meelgroup/approxmc-cert`


28th International Conference on Theory and Applications of Satisfiability Testing (SAT 2025).
Editors: Jeremias Berg and Jakob Nordström; Article No. 32; pp. 32:1–32:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

*Text (Proof Format)*: https://github.com/meelgroup/frat-xor/blob/main/format.md
*Dataset (Benchmark)*: https://doi.org/10.5281/zenodo.15630195

**Funding** This project was supported in part by Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference [RGPIN-2024-05956].
*Yong Kiam Tan*: Singapore NRF Fellowship Programme NRF-NRFF16-2024-0002.
*Magnus O. Myreen*: Swedish Research Council grant 2021-05165.

**Acknowledgements** We sincerely thank the anonymous reviewers for their constructive feedback on our earlier draft of this paper. The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore https://www.nscc.sg, and Niagara supercomputer at the SciNet HPC Consortium. SciNet is funded by Innovation, Science and Economic Development Canada; the Digital Research Alliance of Canada; the Ontario Research Fund: Research Excellence; and the University of Toronto.

## 1 Introduction

Neural networks have had a transformative impact on fields such as image recognition [31], natural language processing [15], and decision making [42], achieving remarkable success in tackling complex tasks in each of those domains. This success has led to their potential deployment in critical scenarios, such as autonomous driving [9], aircraft collision avoidance [29], and drug discovery [50]. However, despite the impressive performance of neural networks, they often exhibit unexpected behaviors and their lack of explainability makes them difficult to control [45]; this has led to significant concerns about their use in high-stakes applications.

Verification techniques for neural networks can help address some of these safety and security concerns [6, 25, 30, 51, 53]. Here, we focus on verification for *binarized neural networks* (BNNs) [26, 27, 38], i.e., where the input/output of every neuron is quantized to a single bit. Our focus is motivated by two practical reasons. First, BNNs are computationally attractive for real-world, resource-constrained use cases – the absence of floating-point arithmetic in BNNs eliminates floating-point issues, and the bit-level representation enables faster inference speeds and more compact memory layouts [23, 26, 37, 40, 58]. Second, existing studies have demonstrated that BNNs can be verified against quantitative specifications, which is beyond the reach of verification methods for more general classes of neural networks [5].

To elaborate on the latter point, the Boolean nature of BNNs allows both the networks and their specifications to be encoded as Boolean formulas which, in turn, enables the use of off-the-shelf SAT solvers and model counters to verify those specifications. For example, prior research has explored both qualitative and quantitative reasoning for BNN robustness, susceptibility to Trojan attacks, and fairness specifications [5, 27, 38, 55]. Unfortunately, existing combinatorial solving methods for such analyses suffer from limited scalability – they apply only to toy-sized BNNs with hundreds of neurons and a few layers. In contrast, custom methods based on abstraction and bounds propagation for verifying input-output specifications scale to much larger (and more general) neural networks [30, 51, 53], but are susceptible to errors; in the annual neural network verification competitions, participating tools have been repeatedly shown to produce incorrect conclusions. *Certified reasoning* [36], where a tool generates both a conclusion and an independently-checkable proof of that conclusion, has long become a mainstay of trustworthiness for modern SAT solvers [24, 32, 46, 52] but it remains nascent for neural network verification tools [53].

In this work, we address the challenges of scalability and trustworthiness in BNN verification by developing an efficient and certified approach for both qualitative and quantitative reasoning on BNNs. Our contributions are as follows.

- For qualitative reasoning, we integrate a native representation of BNN constraints into a modern CNF-XOR SAT solver to enhance its BNN solving efficiency.
- We then extend the solver's associated UNSAT proof format and verified proof checker with native support for BNN constraints, together enabling an efficient solving and certification pipeline for CNF-XOR-BNN formulas.
- Building upon this pipeline, we further develop a certified approximate model counter for quantitative reasoning over BNNs, leveraging native XOR and BNN representations for both effective reasoning and certification.

Empirically, our approach achieves state-of-the-art certified performance in both qualitative and quantitative reasoning for BNNs. Notably, we observe that the compactness of proof certificates with native BNN proof steps enables fast, verified proof checking.

- For qualitative reasoning, our end-to-end approach produced certified answers for 99% of the benchmark UNSAT queries, achieving a $9\times$ speedup over alternative CNF- and PB-based approaches.
- For quantitative reasoning, our certified counting approach answered 86% of the queries, with $218\times$ speedup over the baseline which could only fully certify 4% of the queries.

By developing solving and certification in tandem, our work offers a trustworthy approach to BNN verification with promising scalability. In fact, during the development of our certification pipeline, we identified and fixed a bug in our solver's implementation of the watching scheme for BNN constraints, which highlights the practical importance of certification.

The rest of this paper is organized as follows. Section 2 presents the preliminaries on BNNs used throughout the paper. Section 3 discusses further background and related work. Section 4 introduces the proposed certified solving and counting approaches for BNN verification, and Section 5 evaluates their empirical performance against existing methods. We summarize our findings in Section 6.

## 2 Preliminaries: Binarized Neural Networks

Given a set of Boolean variables $\{x_1, x_2, \ldots, x_n\}$, a *literal* $l$ is either a variable $x$ or its negation $\neg x$. A *clause* $C$ is a disjunction of $k > 0$ literals: $C \coloneqq l_1 \vee l_2 \vee \ldots \vee l_k$. A *Conjunctive Normal Form (CNF) formula* $\varphi$ is a conjunction of $m > 0$ clauses: $\varphi \coloneqq C_1 \wedge C_2 \wedge \ldots \wedge C_m$. A *solution* to $\varphi$ is an assignment of truth values to the variables such that $\varphi$ evaluates to `True`. The formula $\varphi$ is *satisfiable* if at least one solution exists, and *unsatisfiable* otherwise. The *model count* of $\varphi$ refers to the total number of solutions.

Binarized Neural Networks (BNNs) are a class of neural networks in which each neuron is constrained to a Boolean value. Given an input tensor $\mathbf{x} \in \{0,1\}^n$, an output tensor $\mathbf{y} \in \{0,1\}^m$, a weight matrix $\mathbf{w} \in \{-1,1\}^{n \times m}$, and a bias vector $\mathbf{b} \in \mathbb{Z}^m$, a BNN layer maps $\mathbf{x}$ to $\mathbf{y}$ as follows:

$$\mathbf{y} = \mathsf{sign}\left(\mathbf{w}^\top \mathbf{x} + \mathbf{b}\right)$$

Here, the $\mathsf{sign}$ function represents the non-linear sign activation used in BNNs, functioning as a binarized analogue of the ReLU activation. A fully connected BNN consists of several layers laid out in sequence, i.e., the output of one layer is the input of the next, and so on; they can be trained using the straight-through estimator, which enables gradient flow through the sign function during backpropagation. For additional details on BNN training and architecture, we refer the readers to [26].

A BNN constraint corresponds to a neuron in the network, encoding the logical relationship between its input neurons and its output. Formally, we define this relationship as follows:

▶ **Definition 1** (BNN Constraint). *Let $y$ denote the output of a given neuron, and let $x_1, x_2, \dots, x_n$ denote the values of its $n$ input neurons. Let $w_i$ represent the weight associated with the connection from the $i$-th input neuron, and $b$ denote the bias term. Then, the logical input-output relationship of a BNN constraint is defined as:*

$$y \leftrightarrow \sum_{i=1}^{n} w_i x_i + b \geq 0 \tag{1}$$

*where $y$ and $x_i$ are Boolean variables; $w_i$ and $b$ are constants with $w_i \in \{1, -1\}$ for $i \in [1..n]$ and $b \in \mathbb{Z}$.*

We model the value of each binarized neuron as a Boolean variable (in contrast to $y, x_i \in \mathbb{R}$ in conventional neural networks). Some prior work represents neuron values using $\{-1, 1\}$; the two notations are interchangeable via a simple linear transformation. In this work, we adopt the Boolean representation for clarity and compatibility with SAT-based reasoning. The greater-than-or-equal operator in Equation 1 encodes the sign function.

A BNN constraint can be readily transformed into an equivalent *conditional cardinality constraint*, which is defined below. In the rest of this paper, we use the terms BNN constraint and conditional cardinality constraint interchangeably when the context is clear.

▶ **Definition 2** (Conditional Cardinality Constraint). *Given literals $l_1, l_2, \dots, l_n$, an output literal $l_y$, and an integer constant $k$, a conditional cardinality constraint is defined as:*

$$l_y \leftrightarrow \sum_{i=1}^{n} l_i \geq k \tag{2}$$

*where $l_y$ represents the truth value of the cardinality constraint $\sum_{i=1}^{n} l_i \geq k$, and $l_1, l_2, \dots, l_n$ are referred as the left-hand-side (LHS) literals.*

Using the above constraint representations, a BNN with $t$ layers, input bits $\mathbf{l}^0$, and output bits $\mathbf{l}^t$ can be represented by a Boolean formula $\mathrm{BNN}(\mathbf{l}^0, \mathbf{l}^t)$. An input-output specification of the BNN is then a Boolean formula $\mathrm{Prop}\left(\mathbf{l}^0, \mathbf{l}^t\right)$. For instance, a common safety specification is the robustness property, which assesses either the existence (qualitative reasoning) or the number (quantitative reasoning) of adversarial inputs. This property is defined as follows:

▶ **Definition 3** (BNN Robustness). *A BNN is $\varepsilon$-robust for the input $\mathbf{l}^0_g$ and output $\mathbf{l}^t_g$ if there are no satisfying assignments for the following property; $\|\cdot\|_1$ denotes the Hamming distance.*

$$\mathrm{Prop}\left(\mathbf{l}^0, \mathbf{l}^t\right) := \|\mathbf{l}^0 - \mathbf{l}^0_g\|_1 \leq \varepsilon \ \wedge \ \mathbf{l}^t \neq \mathbf{l}^t_g$$

The Boolean nature of BNNs and their property specifications makes it possible to leverage existing solvers and model counters for answering both qualitative and quantitative robustness queries [5, 27, 38, 55]. However, due to the inherent statistical nature of neural network training, the existence of adversarial inputs is expected, making purely qualitative queries potentially uninformative. Quantitatively estimating the prevalence of adversarial inputs allows us to draw statistically significant conclusions [5]. Moreover, the absence of floating-point arithmetic in the formulation of BNNs eliminates floating-point issues when answering these critical robustness queries.

## 3 Related Work

**Neural Network Verification.** Neural network verification has received growing attention, as reflected in the organization of annual benchmark competitions [10]. Existing techniques span several categories, including linear bound propagation (e.g., $\alpha, \beta$-CROWN [51]), reachability analysis (e.g., CORA [1], NNV [35], nnenum [4], NeVer2 [13], and PyRAT [33]), and SMT-based solving (e.g., Marabou [53], NeuralSAT [17], and Reluplex [30]). These approaches focus on qualitative reasoning, particularly the robustness of ReLU networks [22], and often rely on commercial solvers such as Gurobi or MATLAB. However, no fully verified proof checker currently exists for traditional neural network verification frameworks [14]. Another line of research targets the verification of BNNs. This includes qualitative reasoning for robustness [27, 38], quantitative reasoning for robustness, susceptibility to Trojan attacks, and fairness specifications [5, 55], efforts to accelerate verification through increased sparsity in BNNs [27, 39], and techniques for networks with both binarized and non-binarized neurons [2].

**BNN Encodings.** With the conditional cardinality constraint representation, BNN constraints can be further encoded into clauses via cardinality encodings or represented directly as PB constraints. Encoding BNNs into CNF formulas results in a substantial increase in formula size, primarily due to the large number of conditional cardinality constraints. For a BNN with $m$ neurons, where each neuron is represented using Equation 1 with $n$ Boolean variables, the CNF encoding introduces $\mathcal{O}(n)$ variables and clauses per neuron, leading to a total of $\mathcal{O}(nm)$ variables and clauses. As a result, the formula size can easily reach millions of variables and clauses even for networks with only a few hundred neurons, which poses significant challenges for existing SAT solvers and model counters. The complete CNF encoding of BNNs is presented in [5, 38], where BNNs are first encoded as conditional cardinality constraints and subsequently translated into clauses using cardinality encodings. A PB-based encoding for BNNs can be found in [55].

**Propagation and Conflict Analysis for BNN Constraints.** To avoid the cumbersome clause-based encoding of Equation 1, Jia and Rinard introduced an alternative representation of BNN constraints in their solver, MiniSatCS, which enables efficient detection of propagation and conflicts specific to BNNs [27]. MiniSatCS transforms Equation 1 into a reified cardinality constraint, which is logically equivalent to the conditional cardinality constraint in Equation 2. Propagation and conflict detection over Equation 2 are handled as follows.

- **Operand-inferring:** If $l_y$ is assigned `True` and $n-k$ literals among $l_1, \ldots, l_n$ are already assigned `False`, then the remaining unassigned $l_i$ must be assigned `True`; otherwise, a conflict is detected. Conversely, if $l_y$ is assigned `False` and $k-1$ literals among $l_1, \ldots, l_n$ are assigned `True`, then the remaining unassigned literals must be assigned `False`.
- **Target-inferring:** If $k$ literals among $l_1, \ldots, l_n$ are assigned `True`, then $l_y$ must be assigned `True`; otherwise, a conflict is detected. Similarly, if $n-k+1$ literals among $l_1, \ldots, l_n$ are assigned `False`, then $l_y$ must be assigned `False`.

When a conflict is detected, the responsible literals are added to the conflict clause. This design eliminates the need to encode Equation 1 into large sets of clauses, significantly reducing formula size. The native support for direct propagation and conflict detection allows for faster solving and scalability to networks with thousands of neurons. However, reified cardinality constraints are not supported by the UNSAT proof format used in standard SAT solvers, rendering the results uncertifiable without further developments.

**Certified Solving and Counting.** Certification has become a cornerstone of trustworthiness in the SAT community [24, 32, 46, 52]. Recent efforts have extended certification to the CNF-XOR solver, CryptoMiniSat and even to probabilistic counting [49]. The model counting problem aims to determine the number of satisfying assignments for a Boolean formula, while *approximate model counting* seeks to compute a high-quality approximation of this count; approximate model counters have shown practical performance in many applications [5, 16, 21]. The state-of-the-art algorithm, ApproxMC [12, 56], provides a $(1+\varepsilon)$ approximation of the model count with confidence at least $1-\delta$, given tolerance $\varepsilon$ and confidence $\delta$. To ensure correctness, a certification pipeline has been developed for ApproxMC [49], enabling independent verification of its output. In parallel, recent work has also begun to explore *certified exact model counting* [11, 20].

# 4 Certified Reasoning for BNNs

This section presents our integration of native BNN constraints for BNN solving, counting, and proof checking. Section 4.1 introduces our new solver, CryptoMiniSat-BNN, which incorporates native support for BNN constraints to enable efficient qualitative reasoning on BNNs; it also describes our proof format with native support for BNN reasoning in unsatisfiability proofs. Section 4.2 describes our certified model counter, ApproxMCCert-BNN, which similarly leverages native BNN constraint support for efficient and certified quantitative reasoning. Finally, Section 4.3 concludes with a case study of a subtle bug in the watching scheme of CryptoMiniSat-BNN, uncovered by our certification pipeline – this highlights the complementary role of certification in ensuring the correctness of automated reasoning tools.

## 4.1 Certified Solving for Qualitative BNN Reasoning

Our new solver efficiently reasons over both XOR and BNN constraints, collectively referred to as CNF-XOR-BNN formulas; support for XOR constraints is essential for CNF-BNN model counting (which will be presented in Section 4.2). We formally define CNF-XOR-BNN formulas as follows:

▶ **Definition 4** (CNF-XOR-BNN Formula). *Given a set of clauses $C_1, C_2, \ldots, C_m$, XOR constraints $XOR_1, XOR_2, \ldots, XOR_t$, and BNN constraints $BNN_1, BNN_2, \ldots, BNN_s$, a CNF-XOR-BNN formula $\varphi$ is defined as the conjunction of these constraints:*

$$\varphi \coloneqq \bigwedge_{i=1}^{m} C_i \wedge \bigwedge_{j=1}^{t} XOR_j \wedge \bigwedge_{k=1}^{s} BNN_k$$

*A solution (or model) $\omega$ for $\varphi$ is a Boolean assignment to variables that simultaneously satisfies all of the clauses, XORs, and BNN constraints.*

### 4.1.1 Solving CNF-XOR-BNN Formulas

**Input format.** Figure 1 (left) shows an example of a CNF-XOR-BNN formula in our input format, which extends the standard DIMACS format [28] to support XOR and BNN constraints. In the header, the first integer specifies the number of variables, and the second indicates the total number of constraints, including clauses, XORs, and BNNs. Each line beginning with the prefix **b** represents a BNN constraint, while lines starting with the prefix **x** denote XOR constraints. The last line in Figure 1 (left) encodes a BNN constraint corresponding to $x_1 + \neg x_2 + x_3 \geq 2 \leftrightarrow x_4$. The list of integers following the prefix **b**, CNF-XOR-BNN formula

```
p cnf 4 5
1 -2 0
-1 3 0
x 1 -2 -3 0
-4 0
b 1 -2 3 0 2 4 0
```

FRAT-XOR-BNN proof

```
o 1 1 -2 0
o 2 -1 3 0
o x 1 1 -2 -3 0
o 3 -4 0
o b 1 1 -2 3 0 k 2 4 0
i 4 -1 -3 0 b l 1 0 u 3 0
i 5 2 -3 0 b l 1 0 u 3 0
a 6 -3 0 l 1 5 4 0
a 7 -1 0
a 8 -2 0
i 9 1 2 3 0 l 1 0
a 10 0
f 1 1 -2 0
f ...
f 10 0
f x 1 1 -2 -3 0
f b 1 1 -2 3 0 k 2 4 0
```

XLRUP proof

```
o x 1 1 -2 -3 0
i cb 4 -1 -3 0 1 u 3 0
i cb 5 2 -3 0 1 u 3 0
5 d 3 0
6 -3 0 4 1 5 0
6 d 5 4 0
7 -1 0 6 2 0
7 d 2 0
8 -2 0 7 1 0
8 d 1 0
i cx 9 1 2 3 0 1 0
10 0 7 6 9 8 0
```

**Figure 1** Example of a CNF-XOR-BNN formula (left) and its unsatisfiability proof in FRAT-XOR-BNN (middle) and XLRUP (right) formats. Green highlights indicate special keywords, while yellow highlights denote constraint indices.

terminated by a 0, specifies the LHS literals of the cardinality constraint, in this case $x_1, \neg x_2$, and $x_3$. The last two integers, following the terminating 0, indicate the cutoff value and the output literal, respectively. In this example, 2 is the cutoff value, and 4 denotes the output literal $x_4$. Similarly, the line `x 1 -2 -3 0` encodes an XOR constraint: $x_1 \oplus \neg x_2 \oplus \neg x_3 = 1$.

**CNF-XOR-BNN Solving.** A CNF-XOR-BNN solver takes a formula $\varphi$ in the extended DIMACS format and determines its satisfiability. We introduce the first CNF-XOR-BNN solver, CryptoMiniSat-BNN, by extending an existing CNF-XOR solver, CryptoMiniSat [44], with native support for BNN constraints. Our implementation maintains an internal representation of each BNN constraint in the form of a conditional cardinality constraint and directly detects when it causes a unit propagation or conflict. The solver follows the standard Conflict-Driven Clause Learning (CDCL) procedure used in modern SAT solvers while incorporating specialized propagation and conflict detection schemes for XOR and BNN constraints, as outlined in Algorithms 1 and 2.

In Algorithm 1, the unit propagation procedure iteratively processes unit literals stored in trail, starting from the index qhead. For each unit literal $l$, the solver retrieves its watched constraints in Line 4, which may correspond to either a clause or a BNN constraint; XOR watches are maintained separately. The procedure then iterates over all watched constraints in Lines 5–9. If a clause is watched, clausal unit propagation is performed in Line 7. If a BNN constraint is watched, a specialized BNN propagation procedure, PropagateBnn (Algorithm 2), is executed in Line 9. If no conflict is detected during clausal and BNN unit propagation, Gauss-Jordan elimination is applied to propagate the literal $l$ over XOR constraints and to detect potential conflicts (Line 11). Finally, if no conflict is encountered, the index qhead is incremented and the loop continues.

Algorithm 2 describes the procedure, PropagateBnn$(w, l)$, for unit propagation and conflict clause generation from a BNN constraint $w$ with respect to a literal $l$. Let $n$ and $k$ denote the number of LHS literals and the cutoff value of the cardinality constraint in $w$, respectively,

**Algorithm 1** Propagate(qhead, trail).

```
conflict ← empty clause
while qhead < GetSize(trail) and conflict is empty do
    l ← trail[qhead]
    watches ← GetWatches[¬l]
    for w ∈ watches do
        if w is a clause then
            conflict ← PropagateClause(w, l)
        else
            conflict ← PropagateBnn(w, l)
    if conflict is empty then
        conflict ← GaussJordanElim(l)
    qhead ← qhead + 1
return conflict
```

and let $l_y$ denote the output literal. For each BNN constraint, the solver maintains two counters: the number of literals assigned `True` (trueCount) and the number of unassigned literals (undefCount), to enable prompt detection of unit propagation or conflicts. If $l$ is a positive LHS literal in $w$, trueCount is incremented and undefCount is decremented (Lines 7–8); if $l$ is a negative LHS literal, undefCount is decremented (Line 10). Lines 11–24 perform unit propagation in a BNN constraint based on the values of trueCount and undefCount. Specifically, Lines 11–20 handle the propagation of LHS literals. If $l_y$ is `True` and the sum of trueCount and undefCount is less than $k$ (Line 12), a conflict is detected because the cardinality constraint cannot be satisfied even if all unassigned LHS literals are assigned `True`, contradicting the assignment $l_y =$ `True`. This situation arises when at least $n - k + 1$ LHS literals are assigned `False`, causing the sum of trueCount and undefCount to fall below $k$. Consequently, $\neg l_y$ and the $n - k + 1$ `False` LHS literals are added to the conflict clause (Line 13). Alternatively, if the sum of trueCount and undefCount equals $k$, all unassigned LHS literals must be assigned `True` (Line 15) to satisfy the assignment $l_y =$ `True`. Similarly, conflict detection and assignment inference for the case where $l_y$ is assigned `False` are handled in Lines 16–20. On the other hand, Lines 21–24 handle the propagation of the output literal. If trueCount $\geq k$, meaning that at least $k$ LHS literals are assigned `True`, the cardinality constraint in $w$ is satisfied, and the output literal $l_y$ must be assigned `True` (Line 22). Alternatively, if trueCount + undefCount $< k$, it is impossible to satisfy the cardinality constraint, and $l_y$ must be assigned `False` (Line 24).

Whenever a BNN constraint infers the assignment of a literal, a reason clause is generated to facilitate conflict analysis in the CDCL algorithm. For instance, when $l_y$ is inferred to be `True` in Line 22, the reason clause includes $l_y$ and the negation of the $k$ LHS literals assigned to `True`, thereby explaining the inference of $l_y$ based on these assignments.

**Comparison.** CryptoMiniSat-BNN differs from MiniSatCS [27] in several key aspects. First, we maintain BNN constraints in the standard AtLeastK form, as defined in Equation 2, whereas MiniSatCS represents them as AtMostK cardinality constraints. Second, our solver generates minimal conflict and reason clause from BNN constraints, while MiniSatCS does not. For instance, when $l_y$ is assigned `False` and at least $k$ LHS literals are assigned `True`, MiniSatCS includes the negation of all `True` LHS literals in the conflict clause, in addition to $l_y$. In contrast, CryptoMiniSat-BNN adds only the negations of exactly $k$ `True` literals, which

**Algorithm 2** PropagateBnn$(w, l)$.

$n \leftarrow$ GetLiteralCount$(w)$
$k \leftarrow$ GetCutOff$(w)$
$l_y \leftarrow$ GetOutputLiteral$(w)$
trueCount $\leftarrow$ GetTrueCount$(w)$
undefCount $\leftarrow$ GetUndefCount$(w)$
**if** $l$ is a positive literal in $w$ **then**
  trueCount $=$ trueCount $+ 1$
  undefCount $=$ undefCount $- 1$
**else if** $l$ is a negative literal in $w$ **then**
  undefCount $=$ undefCount $- 1$
**if** $l_y$ is `True` **then**
  **if** trueCount $+$ undefCount $< k$ **then**
    add $\neg l_y$ and $n - k + 1$ `False` literals to conflict.
  **else if** trueCount $+$ undefCount $== k$ **then**
    assign `True` to all unassigned literals.
**else if** $l_y$ is `False` **then**
  **if** trueCount $\geq k$ **then**
    add $l_y$ and the negations of $k$ `True` literals to conflict.
  **else if** trueCount $+ 1 == k$ **then**
    assign `False` to all unassigned literals.
**else if** trueCount $\geq k$ **then**
  assign `True` to $l_y$.
**else if** trueCount $+$ undefCount $< k$ **then**
  assign `False` to $l_y$.

constitute the minimal reason for the conflict. Third, CryptoMiniSat-BNN adopts a unified data structure to consistently maintain watches for both clauses and BNN constraints, whereas MiniSatCS employs separate data structures for clause and BNN watches. Lastly, MiniSatCS is built on top of MiniSat, whereas CryptoMiniSat-BNN integrates the BNN constraints directly in the state-of-the-art CNF-XOR solver, CryptoMiniSat, which incorporates more advanced techniques than MiniSat. The native support for both XOR and BNN constraints in CryptoMiniSat-BNN is crucial for our development of a certified CNF-BNN approximate model counter in Section 4.2.

### 4.1.2 Certifying CNF-XOR-BNN Solving

Our new pipeline for certified CNF-XOR-BNN unsatisfiability consists of several parts.

1. We extend the FRAT-XOR[1] format [49] by introducing support for BNN-specific reasoning steps, resulting in an extended FRAT-XOR-BNN format that can be readily generated by CryptoMiniSat-BNN.
2. We develop a format elaborator, FRAT-xor-bnn, which converts FRAT-XOR-BNN proofs into an elaborated proof format, XLRUP (eXtended LRUP). This translator extends FRAT-xor [49], originally designed for FRAT-XOR proofs.

[1] The FRAT-XOR format is itself an extension of the FRAT format [3] with support for XOR-related reasoning steps.

3. The XLRUP proof format supports RUP proofs, extended with both XOR- and BNN-specific steps; we design, implement, and formally verify an efficient proof checker for XLRUP proofs (specifically, we add new BNN reasoning support).

Continuing the example from Figure 1, the unsatisfiability proof for its CNF-XOR-BNN formula in both FRAT-XOR-BNN and XLRUP formats is displayed (middle and right columns, respectively); green highlights indicate special keywords, while yellow highlights denote the constraint indices.

**FRAT-XOR-BNN format.** In the FRAT-XOR-BNN proof format, BNN-related steps are marked with the keyword **b**. The prefix **o b** denotes an original BNN constraint in the input formula, while the prefix **f b** indicates a final BNN constraint that remains after the empty clause is derived. The `clause-from-bnn` step is prefixed with the identifier **i** (denoting implication), followed by the clause implied by a BNN constraint. The keyword **b l** specifies the index of the BNN constraint, while the keyword **u** lists the set of unit clauses used to simply the BNN constraint.

For example, the following step records the derivation of a new clause $(\neg x_1 \vee \neg x_3)$ to be stored at index 4:

```
i 4 -1 -3 0 b l 1 0 u 3 0
```

This clause is implied by the first BNN constraint:

$$x_1 + \neg x_2 + x_3 \geq 2 \leftrightarrow x_4 \tag{3}$$

together with the unit clause $\neg x_4$ at index 3. The chain of unit clauses simplifies the BNN constraint by applying the corresponding variable assignments.

Similarly, the `clause-from-xor` step is prefixed with the identifier **i**, followed by the keyword **l**, which indicates the XOR constraints implying the clause. For instance, the following step records the derivation of a new clause $(x_2 \vee x_1 \vee x_3)$ with index 9:

```
i 9 1 2 3 0 l 1 0
```

This clause is implied by the first XOR constraint:

$$x_1 \oplus \neg x_2 \oplus \neg x_3 = 1 \tag{4}$$

**XLRUP format.** The XLRUP format also marks BNN-specific steps with the keyword **b** and XOR-specific steps with **x**, with slight modifications in notation. Specifically, the keyword (**i cb**) indicates a `clause-from-bnn` step in XLRUP. For example, the following step records the derivation of the clause $(\neg x_1 \vee \neg x_3)$ with index 4, inferred from the first BNN constraint (Equation 3) and the unit clause indexed by 3:

```
i cb 4 -1 -3 0 1 u 3 0
```

Similarly, the keyword (**i cx**) denotes a `clause-from-xor` step in XLRUP. For instance, the following step records the derivation of the clause $(x_1 \vee x_2 \vee x_3)$ with index 9, inferred from the first XOR constraint (Equation 4):

```
i cx 9 1 2 3 0 1 0
```

**Derivation of UNSAT in Figure 1.** We use the FRAT-XOR-BNN proof as an example to illustrate the UNSAT derivation for the CNF-XOR-BNN formula shown in Figure 1. In the proof, after recording the original constraints using the keyword **o**, the first new clause $(\neg x_1 \vee \neg x_3)$ is derived by a `clause-from-bnn` step at index 4. This clause is inferred from the BNN constraint (Equation 3) together with the unit clause $\neg x_4$ indexed by 3. At this step, the solver makes a decision on $x_3$, which triggers propagation of the BNN constraint (Equation 3) as described in Line 20 of Algorithm 2. Specifically, the assignments $\neg x_4$ and $x_3$ imply a `False` assignment to the remaining literals in the BNN constraint. Consequently, $\neg x_3$ (as the reason) and $\neg x_1$ (as the propagated assignment) are added to the reason clause at step 4. Similarly, the reason clause $(x_2 \vee \neg x_3)$ is derived at step 5 from the same BNN constraint to infer a `True` assignment to $x_2$.

Subsequently, $\neg x_1$ and $x_2$ lead to a conflict in the first input clause $(x_1 \vee \neg x_2)$, resulting in the unit learned clause $\neg x_3$ at step 6 through conflict analysis. After backtracking to decision level 0 and applying unit propagation, the assignments $\neg x_1$ and $\neg x_2$ are derived at steps 7 and 8, respectively. Finally, the assignments $\neg x_1, \neg x_2$, and $\neg x_3$ cause a conflict in the XOR constraint (Equation 4), from which the conflict clause $(x_1 \vee x_2 \vee x_3)$ is generated via a `clause-from-xor` step at step 9. The empty clause is then derived at step 10, concluding the proof. The XLRUP proof follows the same reasoning but uses a different syntax.

**Proof generation from CryptoMiniSat-BNN.** CryptoMiniSat-BNN records `clause-from-bnn` steps in a lazy manner (`clause-from-xor` steps are handled similarly). Specifically, the solver logs conflicts and unit propagations from BNN constraints during the search process and generates only the conflict clauses and relevant reason clauses during conflict analysis, which are recorded as `clause-from-bnn` steps. Additionally, we track the list of unit clauses used to simplify BNN constraints during both pre-processing and in-processing, and incorporate them into `clause-from-bnn` steps following the keyword **u**. We disable the variable replacement technique, which replaces a variable in a BNN constraint with another, as it requires reasoning over a BNN constraint and two clauses (or an XOR constraint).[2] We also disable BNN (and XOR) propagation during the distillation phase [18], as our purely clausal RUP proofs do not support BNN propagation.

**FRAT-XOR-BNN to XLRUP through FRAT-xor-bnn.** FRAT-xor-bnn follows the *lightweight* design principle of FRAT-xor [49]: it does not verify the correctness of BNN-specific steps but delegates the checking of those steps to a formally verified tool, cake_xlrup-BNN. Our primary modification involves tracking clauses derived from BNN constraints to ensure their correct usage in subsequent clausal steps and for the automatic elaboration of RUP proofs [3].

**Formally verified proof checking with cake_xlrup-BNN.** Our cake_xlrup-BNN tool extends an earlier proof checker [49] with efficient and verified BNN-specific reasoning. The verification is customized to check `clause-from-bnn` (**i cb**) steps quickly. Briefly speaking, the literals in each BNN constraint are stored in a bitset; whenever a clause $C$ is to be derived, the proof checker tracks propagations for all units from $\neg C$ (and others provided in the proof hint) using the bitset, and accepts the derivation of $C$ if a contradiction is derived. The procedure for checking conflicts is similar to Algorithm 2.

[2] Such replacements can be logged using two clauses, $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$, or an XOR constraint, $(x_1 \oplus x_2 = 0)$ to replace $x_1$ with $x_2$. The clause-based replacement is compatible with clause steps, while the XOR-based replacement is compatible with XOR steps. However, neither is compatible with BNN-specific steps in our proof format.

A particularly useful optimization is to keep the size of each bitset minimal for the corresponding BNN constraint. In straightforward numbering schemes for BNN neurons, the variables appearing in the resulting constraints are dense and contiguous, i.e., $x_i, x_{i+1}, \ldots, x_{i+n}$. In such cases, the corresponding bitset will only allocate memory for the indices $i$ to $i+n$. This optimization ensures that the overall representation takes $O(n)$ space where $n$ is the number of neurons in the BNN, as opposed to $O(nv)$, where $v$ is the total number of variables (if the bitset naively stored indices for every possible variable in the formula).

Note that, by design of the XLRUP extensions, our new BNN steps interact smoothly with the existing proof system for clauses and XOR constraints. As with earlier versions, our extensions are formally verified down to machine code implementations using CakeML [47].

## 4.2 Certified Counting for Quantitative BNN Reasoning

We present our certified model counter, ApproxMCCert-BNN, along with its certificate checker, CertCheck-BNN, designed for efficient quantitative reasoning on BNNs.

### 4.2.1 Model Counting for CNF-BNN Formulas

Let us start by briefly explaining the key idea behind ApproxMC, a state-of-the-art approximate model counting algorithm and implementation [12, 56]. It repeatedly samples and adds random XOR constraints to halve the solution space of the formula; eventually, after adding a number $m$ of such constraints, the number of remaining solutions $n$ will become sufficiently small. Then, the overall solution count is approximated as $n \cdot 2^m$. The core requirement for an efficient implementation is the use of an incremental CNF-XOR solver to find solutions or determine that no further ones exist.

When the input formula $\varphi$ contains both clauses and BNN constraints – that is, when $\varphi$ is a CNF-BNN formula – the same counting algorithm can be used except that a CNF-XOR-BNN solver is required to compute the number of solutions to the combined CNF-XOR-BNN formula $\varphi \wedge XOR_1 \wedge \ldots \wedge XOR_m$.

To this end, we develop ApproxMC-BNN, an approximate model counter built on top of our CNF-XOR-BNN solver, CryptoMiniSat-BNN, introduced earlier. ApproxMC-BNN takes a CNF-BNN formula as input and computes an approximate solution count. The correctness of ApproxMC-BNN is guaranteed by the following theorem; a formalized proof in Isabelle/HOL has established the correctness of ApproxMC for arbitrary Boolean theories [48, 49].

▶ **Theorem 5.** *For a CNF-BNN formula $\varphi$, a tolerance parameter $\varepsilon$, and a confidence parameter $\delta$,* ApproxMC-BNN *returns an approximate count $c$ such that*

$$\mathsf{Pr}\left[\frac{|\mathtt{sol}(\varphi)|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\mathtt{sol}(\varphi)|\right] \geq 1-\delta$$

We emphasize that ApproxMC-BNN is the first model counter that natively supports BNN constraints. This enables it to use a succinct input representation (without extension variables), which significantly improves counting efficiency.

### 4.2.2 Certifying CNF-BNN Counting

The counting certification pipeline for CNF-BNN formulas follows the same framework as the certified CNF model counter, ApproxMCCert, and its certificate checker, CertCheck [49]. Specifically, the certified approximate CNF-BNN model counter, ApproxMCCert-BNN, takes a CNF-BNN formula as input and produces an approximate model count along with a certificate, which can be independently verified by the certificate checker, CertCheck-BNN, to ensure correct execution modulo randomness [49].

Most relevant for us, the design of certificates for CertCheck records the number of XOR constraints used, as well as the solutions to the resulting CNF-XOR-BNN formulas, which are essential for estimating the model count. During certificate checking, CertCheck-BNN verifies the correctness of the solutions to the CNF-XOR-BNN formulas and employs CryptoMiniSat-BNN and cake_xlrup-BNN to generate and validate UNSAT proofs, ensuring exhaustive enumeration of solutions. If all solutions and UNSAT proofs pass verification by CertCheck-BNN, it outputs a certified model count; otherwise, an error is reported [49].

Our new CertCheck-BNN tool was built by extending the earlier formal verification. Thanks to the genericity of the theorems proved in prior work [48], the main verification task here was to develop a theory of CNF-BNN formulas in Isabelle/HOL and then instantiate earlier results to derive a correct-by-construction certificate checking framework, including a formalized version of Theorem 5.

### 4.3 Bug Report from Certification

During the development of our certification pipeline, we identified and resolved a subtle bug in the implementation of CryptoMiniSat-BNN, related to the watching scheme for BNN constraints (Equation 2). As shown in Algorithm 2, CryptoMiniSat-BNN maintains two counters–trueCount (literals assigned `True`) and undefCount (unassigned literals)–to determine the propagation state of a BNN constraint. Each literal is watched according to its polarity: assigning a positive literal increments trueCount and decrements undefCount (Lines 7–8), while assigning a negative literal decrements undefCount (Line 10).

To preserve the AtLeastK form (Equation 2) when the output literal $l_y$ is assigned `False`, CryptoMiniSat-BNN flips the polarity of LHS literals. However, the implementation mistakenly failed to update the polarity of watched literals during this process, resulting in incorrect BNN propagations. This bug was nontrivial to detect through conventional testing but was successfully uncovered by our attempts at running benchmarks through the certification pipeline.

## 5 Experimental Evaluation

In this section, we evaluate the performance of our tools, namely, our proof-generating BNN-based solver (CryptoMiniSat-BNN), proof checker (cake_xlrup-BNN), certified BNN-based model counter (ApproxMCCert-BNN) and its certificate checker (CertCheck-BNN). Experiments were conducted on a BNN robustness benchmark [5], which consists of 960 problem instances with varying BNN architecture sizes.

For comparison, we also transformed the BNN formulas into CNF and PB encodings and evaluated our tools against state-of-the-art certified solvers and counters. Due to the large number of compared tools, we introduce them in their respective subsections.

**Setting.** All experiments were conducted on a high-performance computing cluster, where each node is equipped with an AMD EPYC-Milan processor featuring $2 \times 64$ physical cores and 512GB of RAM. Each solver or counter was required to produce a checkable proof/certificate alongside its output. For every tool, we set a time limit of 500 seconds for qualitative reasoning and 5,000 seconds for quantitative reasoning, with a memory limit of 16 GB. For approximate counting, we used the default parameter values of $\delta = 0.2$ and $\varepsilon = 0.8$.

We report the PAR-2 score for each tool, a standard evaluation metric used in the SAT competition. The PAR-2 score is the weighted average runtime across all instances, including the actual runtime for successfully completed instances, and double the time limit for instances that exceed the timeout (i.e., 1,000 seconds for qualitative reasoning and 10,000 seconds for quantitative reasoning in our setting).

Our experimental evaluation is designed to address the following research questions:

**(RQ1)** How does the certified solving performance of CryptoMiniSat-BNN compare to alternative certified approaches using CNF and PB encodings for BNNs?

**(RQ2)** How does the certified counting performance of ApproxMCCert-BNN compare to the CNF-based baseline for BNNs?

**Summary.** Overall, our approach significantly improves the runtime performance in both qualitative and quantitative reasoning for BNNs, and makes certified BNN verification practically feasible. Specifically,

**(RQ1)** CryptoMiniSat-BNN and cake_xlrup-BNN produced fully certified answers for 99% of the qualitative queries, achieving a 9× speedup over alternative CNF and PB approaches, which fully certified only 62% of all queries.

**(RQ2)** ApproxMCCert-BNN and CertCheck-BNN fully certified results for 86% of the quantitative queries, achieving a 218× speedup over the CNF baseline, which could fully verify only 4% of the queries.

## 5.1 Certified Qualitative Reasoning for BNNs

This section evaluates (RQ1) the runtime performance and proof-checking efficiency of CryptoMiniSat-BNN and cake_xlrup-BNN in answering qualitative queries for BNNs. The baselines include the state-of-the-art CNF solver CaDiCaL [7] (f13d744) with its formally verified LRAT proof checker cake_lpr [46] (36b917a), and the PB solver RoundingSat [19] (73aaf09) with its proof checker VeriPB [8] (178904a).[3] We selected the 125 UNSAT instances (out of 960) as qualitative BNN verification tasks; certifying satisfiability of the remaining instances is straightforward so we reserve those for qualitative reasoning (counting).

A summary performance comparison is shown in Table 1. From the top row, we observe that CryptoMiniSat-BNN successfully solved all 125 instances, achieving the lowest PAR-2 score of 16. The lower row compares the overall performance of solving and proof checking across our three configurations. All but one of the benchmarks were successfully certified by our approach; notably, although RoundingSat solved 123 instances, only 67 proofs were verified by VeriPB within the time limit. This shows the benefit of our native design for both solving and certification.

Figure 2 (left) plots the cumulative number of solved or checked instances for a given time limit, i.e., each point $(x, y)$ indicates that the tool successfully completed $y$ instances within $x$ seconds. CryptoMiniSat-BNN, together with cake_xlrup-BNN, consistently appears at the top of the figure, providing improved solving and checking performance compared to RoundingSat with VeriPB and CaDiCaL with cake_lpr. For solving times, CryptoMiniSat-BNN is on average 26× faster than CaDiCaL and 3× faster than RoundingSat. In fact, CryptoMiniSat-BNN is also 2× faster than MiniSatCS [27]; however, the latter does not support UNSAT proof

[3] The VeriPB toolchain also has a formally verified checker which we did not run for our evaluation. We opted for the static binary of VeriPB from the SAT competition 2023, as our evaluation showed that the 2024 version performed slightly slower on our problem instances.

**Table 1** Runtime performance (time in seconds) for certified qualitative reasoning for BNNs over 125 UNSAT instances. The first row shows solver performance with proof generation, while the second row presents the combined performance of solving and proof checking for various toolchains.

| | CaDiCaL | RoundingSat | CryptoMiniSat-BNN |
|---|---|---|---|
| Solved Instances | 77 | 123 | **125** |
| PAR-2 Score | 478 | 72 | **16** |
| | +cake_lpr | +VeriPB | +cake_xlrup-BNN |
| Solved Instances | 77 | 67 | **124** |
| PAR-2 Score | 874 | 1,068 | **60** |

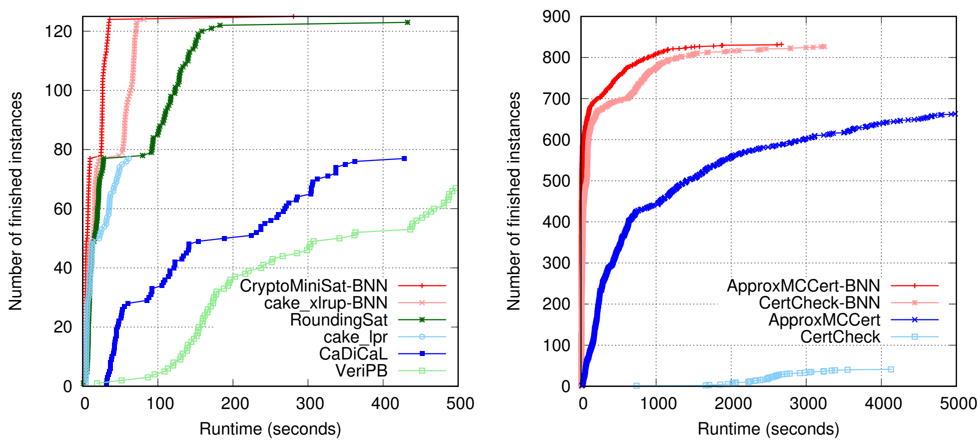


**Figure 2** (Left) Runtime performance comparison of solvers and proof checkers on qualitative reasoning benchmarks. (Right) Runtime performance comparison of counters and certificate checkers. Note that the solving/counting and checking times are plotted separately in both plots.

generation. A detailed comparison with MiniSatCS is presented in Appendix A. For combined solving and checking times, CryptoMiniSat-BNN+cake_xlrup-BNN achieved a 9× speedup over CaDiCaL+cake_lpr and an 18× speedup over RoundingSat+VeriPB. The ratios are calculated over the common solved (and checked) instances for each pair.

## 5.2 Certified Quantitative Reasoning for BNNs

This section evaluates (RQ2) the runtime performance and certificate-checking efficiency of ApproxMCCert-BNN and CertCheck-BNN for quantitative BNN queries. We compare our approach against the certified CNF counter ApproxMCCert and its certificate checker CertCheck [49]. Additionally, we disabled the counting preprocessor Arjun [43] due to its poor performance on BNN benchmarks. Our evaluation is conducted on the full 960 instances of the quantitative robustness benchmark for BNNs [5].

Table 2 presents the runtime performance of the counters and their corresponding certificate checkers. In terms of counting performance (with certificate generation), we observed a significant improvement where ApproxMCCert-BNN solved 169 more instances and more than halved the PAR-2 score to 1,443. For the combined performance of counting and certificate checking, ApproxMCCert together with CertCheck could produce fully certified

**Table 2** Runtime performance (time in seconds) for certified quantitative reasoning for BNNs over 960 counting instances. The first two columns show ApproxMCCert's counting performance, followed by the combined counting and certificate checking performance with CertCheck; the latter two columns similarly show the performance of ApproxMCCert-BNN together with CertCheck-BNN.

| | ApproxMCCert | +CertCheck | ApproxMCCert-BNN | +CertCheck-BNN |
|---|---|---|---|---|
| Finished | 663 | 41 | **832** | **827** |
| PAR-2 Score | 3,778 | 19,256 | **1,443** | **3,068** |

results for only 4% of the instances (41 out of 960) within the time limit. In contrast our approach, ApproxMCCert-BNN+CertCheck-BNN, completed both counting and certificate checking for 86% of the instances (827 out of 960), and substantially reduced the PAR-2 score from 19,256 to 3,068.

Figure 2 (right) shows the number of instances completed by the counters and checkers over time. Here, ApproxMCCert-BNN consistently solved the largest number of instances and it is closely trailed by CertCheck-BNN – thus, ApproxMCCert-BNN offers superior counting runtime and its generated certificates were also readily checked by CertCheck-BNN. In terms of average counting and certification times, ApproxMCCert-BNN+CertCheck-BNN achieved a 218× speedup over ApproxMCCert+CertCheck on their common, fully certified instances.

Upon deeper investigation, a key issue for the certificates of CertCheck is that the CNF encodings of BNN benchmarks lead to *projected* counting instances with an extremely large number of extension variables. The counting certificate format must record satisfying assignments for all variables, which leads to substantial file size, memory, and timing overheads in the CNF-based approach.

In fact, ApproxMCCert-BNN outperforms even existing counters that do not provide certification for quantitative reasoning on BNNs, such as the exact CNF counter Ganak [41], the exact PB counters PBCount [57], and the approximate PB counter ApproxMC-PB [55]. A detailed comparison is provided in Appendix B.

## 6 Conclusion

Certified verification is critical for neural networks deployed in safety-critical applications. This work presents an efficient certified solver for qualitative reasoning and a certified approximate model counter for quantitative reasoning on BNNs. Our approach significantly outperforms existing CNF and PB-based methods and produces fully certified results for the vast majority of queries, making certified BNN verification practically feasible. Looking forward, this framework opens the door to certifying large-scale binarized vision and language models [23, 58], as well as extending certification to quantized neural networks for efficient on-device deployment [34, 54].

## References


1 Matthias Althoff. An introduction to CORA 2015. In Goran Frehse and Matthias Althoff, editors, *ARCH@CPSWeek*, volume 34 of *EPiC Series in Computing*, pages 120–151. EasyChair, 2015. doi:10.29007/ZBKV.

2 Guy Amir, Haoze Wu, Clark W. Barrett, and Guy Katz. An SMT-based approach for verifying binarized neural networks. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *TACAS*, volume 12652 of *LNCS*, pages 203–222. Springer, 2021. doi:10.1007/978-3-030-72013-1_11.

3 Seulkee Baek, Mario Carneiro, and Marijn J. H. Heule. A flexible proof format for SAT solver-elaborator communication. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/LMCS-18(2:3)2022.

4 Stanley Bak. nnenum: Verification of ReLU neural networks with optimized abstraction refinement. In Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, editors, *NFM*, volume 12673 of *LNCS*, pages 19–36. Springer, 2021. doi:10.1007/978-3-030-76384-8_2.

5 Teodora Baluta, Shiqi Shen, Shweta Shinde, Kuldeep S. Meel, and Prateek Saxena. Quantitative verification of neural networks and its security applications. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *CCS*, pages 1249–1264. ACM, 2019. doi:10.1145/3319535.3354245.

6 Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *NIPS*, pages 2613–2621, 2016. URL: https://proceedings.neurips.cc/paper/2016/hash/980ecd059122ce2e50136bda65c25e07-Abstract.html.

7 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *CAV*, volume 14681 of *LNCS*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.

8 Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *J. Artif. Intell. Res.*, 77:1539–1589, 2023. doi:10.1613/JAIR.1.14296.

9 Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. arXiv:1604.07316.

10 Christopher Brix, Stanley Bak, Taylor T. Johnson, and Haoze Wu. The fifth international verification of neural networks competition (VNN-COMP 2024): Summary and results. *CoRR*, abs/2412.19985, 2024. doi:10.48550/arXiv.2412.19985.

11 Randal E. Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn J. H. Heule. Certified knowledge compilation with application to formally verified model counting. *J. Artif. Intell. Res.*, 82, 2025. doi:10.1613/JAIR.1.15958.

12 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable approximate model counter. In Christian Schulte, editor, *CP*, volume 8124 of *LNCS*, pages 200–216. Springer, 2013. doi:10.1007/978-3-642-40627-0_18.

13 Stefano Demarchi, Dario Guidotti, Luca Pulina, and Armando Tacchella. NeVer2: learning and verification of neural networks. *Soft Comput.*, 28(19):11647–11665, 2024. doi:10.1007/S00500-024-09907-5.

14 Remi Desmartin, Omri Isac, Grant O. Passmore, Kathrin Stark, Ekaterina Komendantskaya, and Guy Katz. Towards a certified proof checker for deep neural network verification. In Robert Glück and Bishoksan Kafle, editors, *LOPSTR*, volume 14330 of *LNCS*, pages 198–209. Springer, 2023. doi:10.1007/978-3-031-45784-5_13.

15 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *NAACL-HLT*, pages 4171–4186. Association for Computational Linguistics, 2019. doi:10.18653/V1/N19-1423.

16 Leonardo Dueñas-Osorio, Kuldeep S. Meel, Roger Paredes, and Moshe Y. Vardi. Counting-based reliability estimation for power-transmission grids. In Satinder Singh and Shaul Markovitch, editors, *AAAI*, pages 4488–4494. AAAI Press, 2017. doi:10.1609/AAAI.V31I1.11178.

17 Hai Duong, Dong Xu, ThanhVu Nguyen, and Matthew B. Dwyer. Harnessing neuron stability to improve DNN verification. *Proc. ACM Softw. Eng.*, 1(FSE):859–881, 2024. doi:10.1145/3643765.

18 Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *SAT*, volume 3569 of *LNCS*, pages 61–75. Springer, 2005. doi:10.1007/11499107_5.

19 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-boolean solving. In Jérôme Lang, editor, *IJCAI*, pages 1291–1299. ijcai.org, 2018. doi:10.24963/IJCAI.2018/180.

20 Johannes Klaus Fichte, Markus Hecher, and Valentin Roland. Proofs for propositional model counting. In Kuldeep S. Meel and Ofer Strichman, editors, *SAT*, volume 236 of *LIPIcs*, pages 30:1–30:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SAT.2022.30.

21 Andreas Gittis, Eric Vin, and Daniel J. Fremont. Randomized synthesis for diversity and cost constraints with control improvisation. In Sharon Shoham and Yakir Vizel, editors, *CAV*, volume 13372 of *LNCS*, pages 526–546. Springer, 2022. doi:10.1007/978-3-031-13188-2_26.

22 Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *AISTATS*, volume 15 of *JMLR Proceedings*, pages 315–323. JMLR.org, 2011. URL: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

23 Yefei He, Zhenyu Lou, Luoming Zhang, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. BiViT: Extremely compressed binary vision transformers. In *ICCV*, pages 5628–5640. IEEE, 2023. doi:10.1109/ICCV51070.2023.00520.

24 Marijn Heule, Warren A. Hunt Jr., Matt Kaufmann, and Nathan Wetzler. Efficient, verified checking of propositional proofs. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *ITP*, volume 10499 of *LNCS*, pages 269–284. Springer, 2017. doi:10.1007/978-3-319-66107-0_18.

25 Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In Rupak Majumdar and Viktor Kuncak, editors, *CAV*, volume 10426 of *LNCS*, pages 3–29. Springer, 2017. doi:10.1007/978-3-319-63387-9_1.

26 Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *NIPS*, pages 4107–4115, 2016. URL: https://proceedings.neurips.cc/paper/2016/hash/d8330f857a17c53d217014ee776bfd50-Abstract.html.

27 Kai Jia and Martin C. Rinard. Efficient exact verification of binarized neural networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/1385974ed5904a438616ff7bdb3f7439-Abstract.html.

28 David S. Johnson and Michael A. Trick, editors. *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. DIMACS/AMS, 1996. doi:10.1090/DIMACS/026.

29 Kyle D. Julian, Jessica Lopez, Jeffrey S. Brush, Michael P. Owen, and Mykel J. Kochenderfer. Policy Compression for Aircraft Collision Avoidance Systems. In *DASC*, pages 1–10, 2016.

30 Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kuncak, editors, *CAV*, volume 10426 of *LNCS*, pages 97–117. Springer, 2017. doi:10.1007/978-3-319-63387-9_5.

31 Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *NIPS*, pages 1106–1114, 2012. URL: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

32 Peter Lammich. Efficient verified (UN)SAT certificate checking. *J. Autom. Reason.*, 64(3):513–532, 2020. doi:10.1007/S10817-019-09525-Z.

33 Augustin Lemesle, Julien Lehmann, and Tristan Le Gall. Neural network verification with PyRAT. *CoRR*, abs/2410.23903, 2024. doi:10.48550/arXiv.2410.23903.

34 Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. In Phillip B. Gibbons, Gennady Pekhimenko, and Christopher De Sa, editors, *MLSys*. mlsys.org, 2024. URL: https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html.

35 Diego Manzanas Lopez, Sung Woo Choi, Hoang-Dung Tran, and Taylor T. Johnson. NNV 2.0: The neural network verification tool. In Constantin Enea and Akash Lal, editors, *CAV*, volume 13965 of *LNCS*, pages 397–412. Springer, 2023. doi:10.1007/978-3-031-37703-7_19.

36 Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Comput. Sci. Rev.*, 5(2):119–161, 2011. doi:10.1016/J.COSREV.2010.09.009.

37 Duncan J. M. Moss, Eriko Nurvitadhi, Jaewoong Sim, Asit K. Mishra, Debbie Marr, Suchit Subhaschandra, and Philip Heng Wai Leong. High performance binary neural networks on the xeon+fpga™ platform. In Marco D. Santambrogio, Diana Göhringer, Dirk Stroobandt, Nele Mentens, and Jari Nurmi, editors, *FPL*, pages 1–4. IEEE, 2017. doi:10.23919/FPL.2017.8056823.

38 Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *AAAI*, pages 6615–6624. AAAI Press, 2018. doi:10.1609/AAAI.V32I1.12206.

39 Nina Narodytska, Hongce Zhang, Aarti Gupta, and Toby Walsh. In search for a SAT-friendly binarized neural network architecture. In *ICLR*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=SJx-j64FDr.

40 Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *ECCV*, volume 9908 of *LNCS*, pages 525–542. Springer, 2016. doi:10.1007/978-3-319-46493-0_32.

41 Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A scalable probabilistic exact model counter. In Sarit Kraus, editor, *IJCAI*, pages 1169–1176. ijcai.org, 2019. doi:10.24963/IJCAI.2019/163.

42 David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016. doi:10.1038/NATURE16961.

43 Mate Soos and Kuldeep S. Meel. Arjun: An efficient independent support computation technique and its applications to counting and sampling. In Tulika Mitra, Evangeline F. Y. Young, and Jinjun Xiong, editors, *ICCAD*, pages 71:1–71:9. ACM, 2022. doi:10.1145/3508352.3549406.

44 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *SAT*, volume 5584 of *LNCS*, pages 244–257. Springer, 2009. doi:10.1007/978-3-642-02777-2_24.

45 Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2014. URL: http://arxiv.org/abs/1312.6199.

46 Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen. Verified propagation redundancy and compositional UNSAT checking in CakeML. *Int. J. Softw. Tools Technol. Transf.*, 25(2):167–184, 2023. doi:10.1007/S10009-022-00690-Y.

47 Yong Kiam Tan, Magnus O. Myreen, Ramana Kumar, Anthony C. J. Fox, Scott Owens, and Michael Norrish. The verified CakeML compiler backend. *J. Funct. Program.*, 29:e2, 2019. doi:10.1017/S0956796818000229.

48 Yong Kiam Tan and Jiong Yang. Approximate model counting. *Archive of Formal Proofs*, March 2024. , Formal proof development. URL: https://isa-afp.org/entries/Approximate_Model_Counting.html.

49 Yong Kiam Tan, Jiong Yang, Mate Soos, Magnus O. Myreen, and Kuldeep S. Meel. Formally certified approximate model counting. In Arie Gurfinkel and Vijay Ganesh, editors, *CAV*, volume 14681 of *LNCS*, pages 153–177. Springer, 2024. doi:10.1007/978-3-031-65627-9_8.

50 Bie Verbist, Günter Klambauer, Liesbet Vervoort, Willem Talloen, Ziv Shkedy, Olivier Thas, Andreas Bender, Hinrich W. H. Göhlmann, Sepp Hochreiter, and the QSTAR Consortium. Using transcriptomics to guide lead optimization in drug discovery projects: Lessons learned from the QSTAR project. *Drug Discovery Today*, 20(5):505–513, May 2015.

51 Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *CoRR*, abs/2103.06624, 2021. arXiv:2103.06624.

52 Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *SAT*, volume 8561 of *LNCS*, pages 422–429. Springer, 2014. doi:10.1007/978-3-319-09284-3_31.

53 Haoze Wu, Omri Isac, Aleksandar Zeljic, Teruhiro Tagomori, Matthew L. Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, Pei Huang, Ori Lahav, Min Wu, Min Zhang, Ekaterina Komendantskaya, Guy Katz, and Clark W. Barrett. Marabou 2.0: A versatile formal analyzer of neural networks. In Arie Gurfinkel and Vijay Ganesh, editors, *CAV*, volume 14682 of *LNCS*, pages 249–264. Springer, 2024. doi:10.1007/978-3-031-65630-9_13.

54 Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *ICMS*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR, 2023. URL: https://proceedings.mlr.press/v202/xiao23c.html.

55 Jiong Yang and Kuldeep S. Meel. Engineering an efficient PB-XOR solver. In Laurent D. Michel, editor, *CP*, volume 210 of *LIPIcs*, pages 58:1–58:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CP.2021.58.

56 Jiong Yang and Kuldeep S. Meel. Rounding meets approximate model counting. In Constantin Enea and Akash Lal, editors, *CAV*, volume 13965 of *LNCS*, pages 132–162. Springer, 2023. doi:10.1007/978-3-031-37703-7_7.

57 Suwei Yang and Kuldeep S. Meel. Engineering an exact pseudo-Boolean model counter. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *AAAI*, pages 8200–8208. AAAI Press, 2024. doi:10.1609/AAAI.V38I8.28660.

58 Yichi Zhang, Ankush Garg, Yuan Cao, Lukasz Lew, Behrooz Ghorbani, Zhiru Zhang, and Orhan Firat. Binarized neural machine translation. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *NeurIPS*, 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/bd1fc5cbedfe4d90d0ac2d23966fa27e-Abstract-Conference.html.

**Table 3** Runtime performance comparison of CryptoMiniSat-BNN and MiniSatCS over 960 instances.

| | MiniSatCS | CryptoMiniSat-BNN |
|---|---|---|
| Solved Instances | 960 | 960 |
| PAR-2 Score | 3 | 1 |

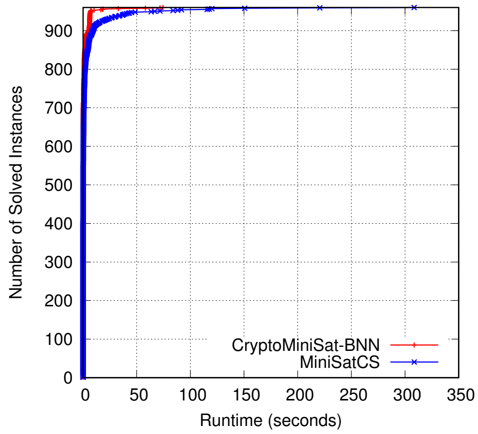


**Figure 3** Runtime performance comparison of CryptoMiniSat-BNN and MiniSatCS.

## A Solving Performance Comparison with MiniSatCS

We present a performance comparison between CryptoMiniSat-BNN and MiniSatCS [27], excluding proof generation, which is not supported by MiniSatCS. Our evaluation is conducted on the full set of 960 instances from the quantitative robustness benchmark for BNNs [5], including both SAT and UNSAT cases. We set a time limit of 500 seconds and a memory limit of 4GB for each instance.

As shown in Table 3, both CryptoMiniSat-BNN and MiniSatCS solved all instances. However, CryptoMiniSat-BNN achieved a lower PAR-2 score of 1, compared to 3 for MiniSatCS. Figure 3 illustrates the number of solved instances over time. CryptoMiniSat-BNN solved all instances within 75 seconds, whereas MiniSatCS required up to 309 seconds. Overall, CryptoMiniSat-BNN achieved a 2× speedup over MiniSatCS.

## B Counting Performance Comparison

We present the counting performance comparison between ApproxMC-BNN and state-of-the-art CNF counters, Ganak [41] and ApproxMC [56], and PB counters, PBCount [57] and ApproxMC-PB [55] in Table 4; ApproxMC-BNN refers to ApproxMCCert-BNN without certification generation. We also repeat the numbers for ApproxMCCert-BNN for comparison. Each counter uses a memory limit of 4GB.

The CNF counters demonstrated the worst performance. The exact and approximate counter, Ganak and ApproxMC solved only 424 and 770 out of 960 instances, respectively, with the two highest two PAR-2 scores in the table. The approximate PB counter, ApproxMC-PB solved 816 instances and lowered the PAR-2 score to 1,741. PBCount failed to solve any instance so we removed it from the table. Lastly, ApproxMC-BNN achieved the best perform-

**Table 4** Counting performance comparison over 960 instances.

| | Ganak | ApproxMC | ApproxMC-PB | ApproxMC-BNN | ApproxMCCert-BNN |
|---|---|---|---|---|---|
| Counted Instances | 424 | 770 | 816 | **868** | 832 |
| PAR-2 Score | 6248 | 2600 | 1741 | **1128** | 1,443 |

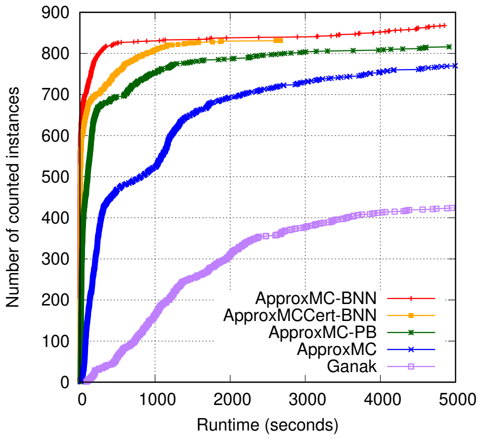


**Figure 4** Runtime performance comparison between counters.

ance with 868 instances solved and further reduced the PAR-2 score to 1,128. ApproxMC-BNN solved 98 more instances than the best CNF counter (ApproxMC) and outperformed the best PB counter ApproxMC-PB by 52 more solved instances. Even with the overhead of certificate generation, ApproxMCCert-BNN still outperforms other non-BNN-native approaches.

Figure 4 compares the counting performance in terms of counted instances per runtime. The plot shows that ApproxMC-BNN consistently solves the most instances at any time limit, followed by ApproxMC-PB, ApproxMC, and Ganak.