Truth Table Net: Scalable, Compact & Verifiable Neural Networks with a Dual Convolutional Small Boolean Circuit Networks Form

Adrien Benamira[1] , **Thomas Peyrin**[1] , **Trevor Yap**[1] , **Tristan Guérand** [1] , **Bryan Hooi** [2]
[1]Nanyang Technological University
[2]National University of Singapore
{adrien.benamira, thomas.peyrin, trevor.yap}@ntu.edu.sg, guer0001@e.ntu.edu.sg, bhooi@comp.nus.edu.sg

Abstract

We introduce "Truth Table net" (`TTnet`), a novel Deep Neural Network (DNN) architecture designed to provide excellent scalability/compactness trade-offs among DNNs, allowing in turn to tackle the DNN challenge of fast formal verification. `TTnet` is constructed using Learning Truth Table (LTT) filters, analogous to how a Deep Convolutional Neural Network (DCNN) is built upon convolutional filters. The differentiable LTT filters are unique by their dual form: they are both a neural network-based function and a small-sized truth table that can be computed within a practical time frame. This characteristic guarantees, by design and independently of the overall architecture, the ability to practically extract an efficient (in terms of the number of logical gates) and functionally equivalent Conjunctive Normal Form (CNF) Boolean logic gate implementation. This CNF circuit is even *optimal* when the LTT truth table's input bit size $n \leq 12$. In particular, `TTnet` architecture is the first differentiable DNN with as dual form a compact logic gate representation that can scale to datasets larger than CIFAR-10: we achieve an accuracy of 41% on the ImageNet dataset while ensuring that each LTT filter truth table is fully computable within 2^{16} operations. We further compare the compactness and scalability performances of `TTnet` Boolean logic circuit representation to state-of-the-art differentiable logic DNNs across tabular, MNIST, and CIFAR-10 datasets. We emphasize that `TTnet` is the first solution to the open problem of designing differentiable *convolutional* neural networks with an exact dual logic gate circuit representation, bridging the gap between symbolic AI and trainable DCNNs. Finally, as improving DNNs compactness in Boolean logic circuit form reduces the complexity of their formal verification, we demonstrate `TTnet` effectiveness in exact sound and complete formal verification. Notably, our model achieves robustness verification in $\approx 10ms$ vs $\approx 100s$ for traditional state-of-the-art DNNs solvers.

1 Introduction

DNNs' success in various machine learning tasks has been remarkable [Goodfellow *et al.*, 2016], but their opacity has raised concerns about their security [AI, 2023; Commission, 2021], necessitating the development of formal verification processes to ensure safety and reliability [Driscoll, 2020]. Formal verification employs mathematical techniques to prove that a system satisfies predefined properties; however, the complexity of DNNs poses significant challenges for verification [Wang *et al.*, 2021; Brix *et al.*, 2023]. Traditional DNNs struggle to simultaneously achieve scalability, verifiability, and compactness in terms of logic gates [Petersen *et al.*, 2022]. Ideally, a natural goal is to design a DNN that can be transformed into a compact Boolean logic circuit without sacrificing high accuracy on large-scale datasets. This approach holds the potential to enhance the compactness of DNNs and facilitate their verification processes [Jia and Rinard, 2020].

Our approach. DNNs, similar to ciphers, represent complex black-box functions that must be trustworthy. Drawing inspiration from a popular design principle of symmetric-key encryption algorithms (so-called Substitution-Permutation Networks or SPN [Daemen and Rijmen, 2002]), our paper proposes compact and small learnable filter Convolutional Neural Networks (CNNs) equivalent by design into small truth tables (from 6 to 16 bits of input). To achieve that goal, we adopt a divide-and-conquer strategy. In the divide phase, before training, we reduce the input size of each DNN building block function, enabling the computation of its complete distribution within a practical time, independent of its overall architecture. In the conquer phase, after training, we compute the complete distribution of each building function.

Therefore, our objective is to design DNNs based on functions that allow for full distribution computation within a practical timeframe, independent of the overall architecture, all while maintaining accurate performance on large datasets.

1.1 Our Contributions And Claims

The LTT filter. To address this challenge, our paper introduces a novel filter function called the Learning Truth Table (LTT) filter. The LTT filter serves as the fundamental building block function of `TTnet`, similar to how a CNN filter is utilized in a DCNN. The LTT filter is defined based on three key rules and presents six attractive properties, see Section 4.1.

The `TTnet` architecture. LTT layers are assembled into a `TTnet` in the same way that CNN filters are assembled into DCNNs. Specifically, LTT filters are stacked together in LTT layers, and these LTT layers are stacked together in the overall `TTnet`. The final classification layer is linear.

Claim on scalability. In contrast to previous differentiable logic DNN solutions, `TTnet` demonstrates remarkable scalability, achieving performance comparable to other DNNs based on Boolean functions on the ImageNet classification task. For a fair comparison, we specifically evaluated `TTnet` against XnorNet [Rastegari *et al.*, 2016] and the Binary Neural Networks (BNNs) original paper [Hubara *et al.*, 2016], both of which are also based on Boolean functions. The results showcase that `TTnet`, which is a differentiable convolutional logic gate DCNN as well as a Boolean function, can effectively scale to large-scale datasets like ImageNet, achieving competitive performance **(claim 1)**.

Claims on compactness. `TTnet` stands as the pioneering differentiable DCNN that can be transformed into a Boolean logic circuit by design **(claim 2)**, which was already identified as an open problem [Petersen *et al.*, 2022]. Empirical evaluation on small datasets like tabular datasets (Adult and Breast Cancer) and MNIST reveals that `TTnet` is equivalent to state-of-the-art differentiable logic gate DNN (Diff Logic Net [Petersen *et al.*, 2022]) in terms of compactness and accuracy **(claim 2A)**. Furthermore, at the cost of higher complexity, `TTnet` exhibits superior accuracy and training time scalability, achieving a notable accuracy of 70.75% on CIFAR-10 with less than an hour of training, marking an $+8\%$ improvement and $\times 90$ faster training compared to Diff Logic Net [Petersen *et al.*, 2022] **(claim 2B)**.

Claims on formal verification. As a demonstration of its excellent scalability/compactness profile, we show that `TTnet` is very well suited to support fast property verification using generic SAT solvers **(claim 3)**. This allows for the formal verification of the network's robustness in dozens of milliseconds on MNIST and CIFAR-10, which is about 10^4 times faster than $\alpha - \beta$-Crown [Xu *et al.*, 2020; Wang *et al.*, 2021], the state-of-the-art DNN verification solver and a recent winner of the Verification Neural Network (VNN) competition [Brix *et al.*, 2023] **(claim 3A)**. We also provide a comparison with BNNs verification [Jia and Rinard, 2020; Narodytska *et al.*, 2019]: our method is more robust in low-noise cases and faster to verify in all scenarios, for MNIST/CIFAR-10 **(claim 3B)**.

Outline. Section 2 presents a literature review on Boolean logic gates DNN and DNN formal verification. Section 3 provides a comprehensive overview of the background in the field of Boolean logic and formal verification. Section 4 describes the design principles and architecture of the `TTnet` model. In Section 5, we provide a thorough evaluation of the performance of our model on various datasets, carefully demonstrating the above claims. Finally, we discuss the limitations and opportunities in Section 6, and conclude in Section 7.

2 Related Works

Binary and Sparse Neural Networks. BNNs are DNN architectures where the activations and weights within a neural network use binary states, typically representing $\{-1, +1\}$. This innovative technique allows for the approximation of computationally expensive matrix multiplications through faster XNOR and bitcount (popcount) operations. BNNs are characterized primarily by their weights. Sparse Neural Networks (SNNs) represent another approach to neural network design. In SNNs, only a subset of connections is present, in contrast to fully-connected layers. In the SNNs literature, the primary focus often revolves around distilling a sparse neural network from a dense one, with careful consideration given to the choice of connections. However, recent research has suggested the high efficacy of using randomized and fixed sparse connections from the outset. In our experiments, we include BNNs and SNNs as baseline models, due to their exceptional inference speed performance.

DNNs into compact logic gate circuit representation. [Chatterjee, 2018] proposed a DNN based on truth tables, but it did not scale beyond MNIST. Other works aimed to convert DNNs into compact Boolean logic circuit designs, which are essential for deploying DNNs on resource-constrained devices [Wang *et al.*, 2019]. However, the challenge in learning logic gate networks is that they are typically non-differentiable, making them difficult to train with gradient descent [Rumelhart *et al.*, 1986]. To date, there is no family of differentiable DCNNs logic gates, as stated by [Petersen *et al.*, 2022]and our LTT filter function is proposed as a possible bridge to that gap between symbolic AI and trainable DCNNs.

Exact, complete, and sound formal verification. Formal verification is a critical aspect in ensuring the DNN's soundness and completeness properties, particularly for safety-critical applications [Driscoll, 2020]. Conventional verifiers work with real-valued networks, but they face scalability challenges, e.g. hundreds of seconds to verify an MNIST image property [Müller *et al.*, 2022], and provide no guarantees of correctness due to floating point errors [Jia and Rinard, 2021]. Therefore, developing an efficient DNNs verification process is an important topic, as evidenced by the $\alpha - \beta$-Crown paper [Xu *et al.*, 2020; Wang *et al.*, 2021] and winner of the VNN competition [Brix *et al.*, 2023]. In the present work, instead of designing a new general verification method for DNNs, we propose the first architecture that can be efficiently and correctly verified with **any** SAT solver [Roussel and Manquinho, 2009]. This research direction is attracting interest, as exemplified by [Jia and Rinard, 2020], who strives to show that BNNs [Hubara *et al.*, 2016] are faster to verify if a specific well-crafted SAT solver is designed. Further discussion regarding incomplete methods is given in Appendix.

Local features for accurate DNN. Recent works by [Brendel and Bethge, 2018] and [Agarwal *et al.*, 2021] experimentally showed that highly nonlinear functions (Resnet for the former, expanding autoencoder for the latter) can enable a DNN to learn high-quality local features (7×7 floating input pixels on image dataset for the former, 1×1 floating input on tabular dataset for the latter). In this work, we build upon the idea of using highly nonlinear functions to learn local features.

However, instead of processing local-dimensional floating-point inputs, `TTnet` reduces the local features to binary inputs ($n \leq 16$). These two approaches legitimize our use of `TTnet` on tabular and image datasets and highlight the importance of Rule 3 in Section 4.1.

3 Background

Boolean logic. A truth table is a mathematical representation of the output of a Boolean function for all possible input combinations. It is usable in practice when the corresponding Boolean function distribution is fully computable. Note that the Boolean function distribution of a DNN/BNN is not fully computable, as the input size of the function is too large. A truth table can be expressed in CNF[1], being the standard form to represent a Boolean function, especially in formal verification. A Boolean logic circuit is a physical or mathematical representation of a Boolean function, made up of interconnected logic gates [Arora and Barak, 2009; Klir *et al.*, 1997], typically AND/OR gates.

Formal verification. Our framework is built upon [Narodytska *et al.*, 2019], a theoretical exact sound and complete framework to verify any property. Given a precondition $prec$ on inputs x, the property $prop$, on outputs o, and the SAT relations provided by a DNN between inputs and outputs denoted as $DNN(x, o)$, we assess the validity of the statement $prec(x) \wedge DNN(x, o) \implies prop(o)$. To show the existence of a counter-example to this robustness property, we search for a satisfying assignment of $prec(x) \wedge DNN(x, o) \wedge \overline{prop(o)}$ (Eq1) with a proven SAT solver. An "unsat" result from the SAT solver proves no noise exists for satisfying (Eq1) and therefore no noise can attack; on the opposite, a "sat" answer proves that there is one noise that satisfies (Eq1) to attack the DNN. In that case, the noise is given by the SAT solver and then tested for validation. In this paper, one distinguishes the traditional *natural accuracy* from the *verified accuracy*, the latter measuring the fraction of predictions that remain correct for all adversarial attacks within the perturbation constraints.

4 Truth Table Neural Networks

First, we define LTT filters, give their properties, and illustrate their use with an example in Section 4.1. We then explain how these LTT filters are integrated into `TTnet` in Section 4.2. We also provide a companion video in Supplementary Material.

4.1 Overall LTT Filter Design

General Design Criteria And Rules Of LTT Filters.
We aim to develop a novel filter function, called the Learning Truth Table (LTT) filter, that is comparable to traditional CNN filters in terms of computational efficiency[2] but differs in complexity. Specifically, we design an LTT filter such that one of its main characteristics is to be a grouped CNN filter: it will be sparser in connectivity than a classical CNN filter. The LTT filter essential criteria are:
(A) The LTT filter distribution must be entirely computable in practical time, independently of the overall DNN architecture.
(B) Once LTT filters are assembled into a layer and layers into a DNN, the latter should be scalable, especially on large datasets such as ImageNet.

To achieve that, we define three LTT filter design rules:

Rule 1: Force the input bit size n of the LTT filter to be $n \leq 16$, independently of the architecture.

Rule 2: Use binary inputs/outputs, but with real-valued weights and intermediate values.

Rule 3: Ensure that the LTT filter uses nonlinear functions in between the Heaviside activations.

As a result, each filter in our architecture becomes a truth table with a maximum input bit size of 16, leading to a compact DNN represented as a Boolean logic gate circuit that is easy to train and verify. We explore these benefits in Section 5.

Regular 2D-Convolution Filters.
We first define below a traditional 2D-Convolution filter.

Definition 1. Let an input $X \in \mathbb{R}^{C_{\text{in}} \times H \times W}$ where C_{in}, H, W represent the number of channels, height, and width of a channel respectively. If a regular convolution filter is applied on X with kernel size $k \times k$, group G, stride $s = 1$ padding $p = 0$, the output is denoted as $O \in \mathbb{R}^{C_{\text{out}} \times H \times W}$ where every output unit $o_{ij} \in \mathbb{R}^{C_{\text{out}}}$ is:

$$o_{ij} = o^1_{i,j} \cup \cdots \cup o^G_{i,j} \tag{1}$$

with $\cup$ the concatenation operation and unit $o^\gamma_{ij} \in \mathbb{R}^{C_{\text{out}}/G}$ representing the output in group γ at position i, j. Namely:

$$o^\gamma_{i,j} = \Psi_{\omega^\gamma}(x^\gamma_{i,j}, \ldots, x^\gamma_{i+k-1,j+k-1}) = \sum_{u=0}^{k-1}\sum_{v=0}^{k-1} x^\gamma_{(i+u,j+v)}\omega^\gamma_{u,v} \tag{2}$$

where $i \in \{1, \ldots, H\}, j \in \{1, \ldots, W\}, \gamma \in \{1, \ldots, G\}$, with the input $x^\gamma_{(i+u,j+v)} \in \mathbb{R}^{C_{\text{in}}/G}$ and the convolution filter weights $\omega^\gamma_{u,v} \in \mathbb{R}^{(C_{\text{in}}/G) \times (C_{\text{out}}/G)}$.

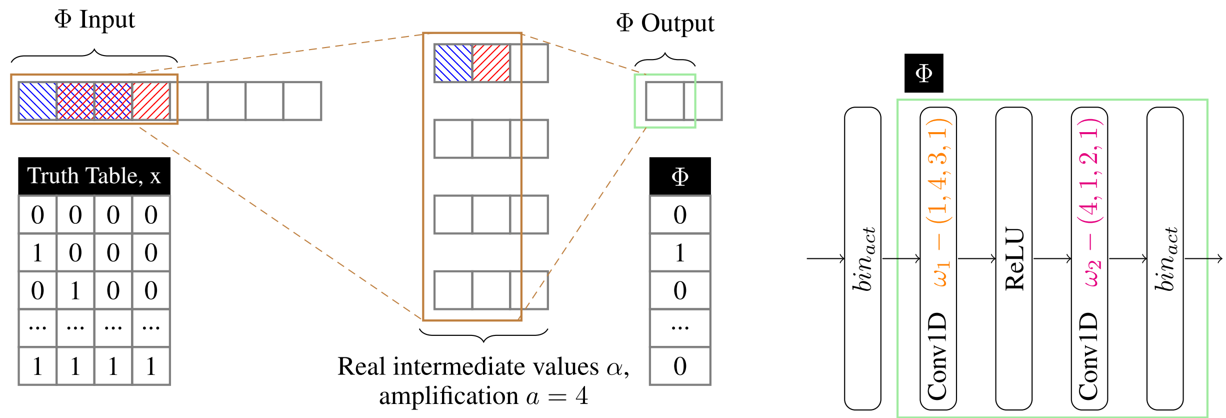
Property 2. Given that the set of inputs $(x^\gamma_{i,j}, \ldots, x^\gamma_{i+k-1,j+k-1})$ is encoded on q bits, the complete distribution of one 2D-CNN filter of function Ψ_{ω^γ} can be computed in $2^{q \times k \times k \times (C_{\text{in}}/G)}$ operations. Furthermore, the weights ω^γ are real and learnable using gradient descent.

Definition of LTT filters. We aim to significantly reduce the complexity of the 2D-CNN filter, by designing a function whose input bit size $q \times k \times k \times (C_{\text{in}}/G)$ is smaller than 16. We now define LTT filters as below.

Definition 3. Let an input $X \in \{0, 1\}^{C_{\text{in}} \times H \times W}$ where C_{in}, H, W represent the number of channels, height, and width of a channel respectively. Let the group and the kernel size parameters be such that $G = C_{\text{in}}/n$ and $k^2 \leq 16/n$, $n \in \{1, 4, 16\}$). We define two convolutions filter weights $\omega^\gamma_1 \in \mathbb{R}^{k_1 \times k_1 \times (C_{\text{in}}/G) \times (a \times C_{\text{out}}/G)}$ and $\omega^\gamma_2 \in \mathbb{R}^{k_2 \times k_2 \times (a \times C_{\text{out}}/G) \times (C_{\text{out}}/G)}$ with both stride 1, no padding

[1]CNF is a conjunction of disjunctions of literals, for example $\Omega = (c_1 \wedge \cdots \wedge c_m)$, with each clause $c_j = (l_{j1} \vee \cdots \vee l_{jr})$ and where $(l_{j1}, \ldots, l_{jr})$ are Boolean literals.

[2]We define the computational efficiency in binary operations, denoted as OPs, as the total number of binary gates (NAND/NOR/AND/OR/XOR/XNOR) required to represent the entire DNN in Boolean logic gate format; and the floating-point operations counterpart denoted as FLOPs



(a) LTT filter Φ computation and transformation into a truth table. Φ is characterized by weights ω_1 with parameters (input channel, output channel, kernel size, stride) = $(1, 4, 3, 1)$, and $\omega_2 = (4, 1, 2, 1)$. Φ input bit size is 4 (i.e., brown box) since the output feature (i.e., green box) requires 4 input entries (i.e., blue & red hashed boxed). Equation (3) gives a formal definition.

(b) LTT overview in one dimension. The intermediate values and the weights (ω_1, ω_2) are real and the input/output values are binary. bin_{act}: Heaviside step function defined as: $bin_{act}(x) = (1 + sgn(x))/2$ with $x \in \mathbb{R}$.

Figure 1: A Learning Truth Table (LTT) filter example in one dimension.

and such that $k_1 + k_2 - 1 = k$ and a is the amplification ratio. We also define Θ to be a nonlinear function. The LTT output is denoted as $O \in \mathbb{R}^{C_{\text{out}} \times H \times W}$ where every output unit $o_{ij} \in \mathbb{R}^{C_{\text{out}}}$ is defined as in Equation (1). Unit $o_{ij}^{\gamma} \in \mathbb{R}^{C_{\text{out}}/G}$, the output in group γ at position i, j is given by

$$\begin{aligned} o_{i,j}^{\gamma} &= \Phi_{\omega_1^{\gamma}, \omega_2^{\gamma}}(x_{i,j}^{\gamma}, \ldots, x_{i+k-1,j+k-1}^{\gamma}) \\ &= \text{bin}_{\text{act}}(\Psi_{\omega_2^{\gamma}}(\alpha_{i,j}^{\gamma}, \ldots, \alpha_{i+k_2,j+k_2}^{\gamma})) \end{aligned} \tag{3}$$

with $\Psi_{\omega^{\gamma}}$ defined as in Equation (2) and $\alpha_{i,j}^{\gamma} \in \mathbb{R}^{a \times C_{\text{out}}/G}$ as

$$\alpha_{i,j}^{\gamma} = \Theta(\Psi_{\omega_1^{\gamma}}(x_{i,j}^{\gamma}, \ldots, x_{i+k_1-1,j+k_1-1}^{\gamma}))$$

where $i \in \{1, \ldots, H\}, j \in \{1, \ldots, W\}, \gamma \in \{1, \ldots, G\}$ and with the input $x_{(i,j)}^{\gamma} \in \mathbb{R}^{C_{\text{in}}/G}$.

By definition, the described LTT filters validate the three rules given previously. In Figure 1a, we provide an example of an LTT computation in one dimension for $k_1 = 3, k_2 = 2, k = k_1 + k_2 - 1 = 4, a = 4, C_{\text{in}} = 1, C_{\text{out}} = 1$ and $\Theta = \text{ReLU}$. The architecture is depicted in Figure 1b.

LTT Filters Properties.
We point six main LTT properties. Properties **P1** and **P2** of the proposed `TTnet` architecture offers practical benefits, including fast and scalable filter training. Meanwhile, properties **P3**, **P4**, **P6** lead to the development of compact LTT filters and efficient verification techniques. Figure 2 provides an example of the two forms of **P5**.

Example: From LTT Weights To Truth Table To CNF.
Consider a trained 1D-LTT Φ_{ω_1,ω_2} with input size $n = 4$, a stride of size 1, and no padding. The architecture of Φ_{ω_1,ω_2}, given in Figure 1b, is composed of two CNN filter layers: the first one has parameters ω_1 with (input channel, output channel, kernel size, stride) = $(1, 4, 3, 1)$, while the second $\omega_2 = (4, 1, 2, 1)$. The values of the weights (ω_1, ω_2) are given in Figure 2. The inputs and outputs of Φ_{ω_1,ω_2} are binary, and we denote the inputs as $[x_0, x_1, x_2, x_3]$. To compute the entire distribution of Φ_{ω_1,ω_2}, we generate all $2^4 = 16$ possible input/output pairs, as shown in Figure 1a, and obtain the truth table in Figure 2. This truth table fully characterizes the behavior of Φ_{ω_1,ω_2}. We then transform the truth table into an optimal CNF using the Quine-McCluskey algorithm [Blake, 1938; Udovenko, 2023]. This optimal CNF fully characterizes the behaviour of Φ_{ω_1,ω_2} as well and is exactly equivalent.

- **P1:** The LTT filter weights (ω_1, ω_2) are trainable with gradient descent and the Straight-Through Estimator (STE) to handle the input/output binarisation.
- **P2:** The LTT filter preserves real-valued weights (ω_1, ω_2) and intermediate values α.
- **P3:** The entire distribution of the LTT filter can be calculated in $2^n \leq 2^{16} = 65,536$ operations (less than 1 ms on a standard PC). Truth table input bit size n is independent of the architecture.
- **P4:** The Quine-McCluskey algorithm [Blake, 1938] can be used to compute the **optimal** CNF (in terms of Boolean logic gates) from the LTT truth table if $n \leq 12$. For $12 < n \leq 16$ a compact CNF can be computed [Udovenko, 2023].
- **P5:** The LTT filter has two forms: neural network weights (ω_1, ω_2), or a small Boolean circuit (as a truth table or as Boolean logic gates).
- **P6:** Under truth table or CNF form, an LTT evaluation does not need to compute any activation function.

4.2 Overall TTnet Design

We integrated LTT filters into the neural network, just as CNN filters are integrated into a deep convolutional neural network: each LTT layer is composed of multiple LTT filters and there are multiple LTT layers in total (see Figure 2). Additionally,

$$\omega_1 = \begin{pmatrix} 10 & -1 & 3 \\ 6 & -5 & 4 \\ 4 & 4 & -3 \\ 4 & 4 & 3 \end{pmatrix}$$

$$\omega_2 = \begin{pmatrix} -5 & 0 & 9 & -5 \\ -5 & 4 & 0 & 0 \end{pmatrix}$$

x_0	x_1	x_2	x_3	Φ
0	0	0	0	0
0	0	0	1	1
0	0	1	*	0
0	1	*	*	0
1	*	*	*	0

CNF: $x_3 \wedge \overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2}$

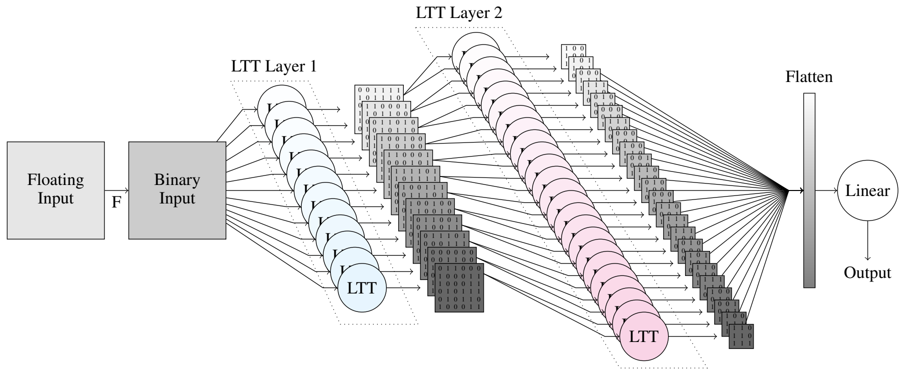


Figure 2: On the left, we present the two forms of 1D-LTT Φ given in Figure 1: on top the DNN weights (ω_1, ω_2), on bottom the truth table / (optimal) CNF. The truth table uses * to denote both 0/1 possible values. On the right, we present the general architecture of `TTnet` with one-channel input. The image undergoes binarization through processing F, followed by two LTT layers, flatten and linear layer.

Accuracy	$\texttt{TTnet}_{16-\|8}$	Original BNN	XnorNet
top 1	41.6 % ± 0.6	27.9 %	44.2 %
top 5	65.1 % ± 0.7	50.4 %	69.2 %

Table 1: Comparison of top 1 and top 5 natural accuracy on ImageNet (experiment has been repeated three times with different seeds).

n	24	20	16	12	8	4
Acc.	89.1% ± 0.2	87.8% ± 0.2	86.0% ± 0.3	84.3% ± 0.2	81.2% ± 0.4	77.5% ± 0.4

Table 2: Ablation study of $\texttt{TTnet}_{n-|8}$ for different n on CIFAR-10. Acc. stands for accuracy.

there is a pre-processing layer and a final layer, which provide flexibility in adapting to different applications: scalability, formal verification, and logic circuit design.

5 Results

Experimental environment. The project code can be found in Supplementary Material and was coded in Python with PyTorch library [Paszke *et al.*, 2019] for training, Numpy [Van Der Walt *et al.*, 2011] for testing as we infer with truth tables. We used 4 Nvidia GeForce RTX 3090 GPUs and 8 cores Intel(R) Core(TM) i7-8650U CPU clocked at 1.90 GHz, 16 GB RAM. We note $\texttt{TTnet}_{n-k}$ a `TTnet` with truth tables of input bit size n, without pre-processing layer and a final linear layer with weights quantized on k bits; and $\texttt{TTnet}_{n-|k}$, with a pre-processing layer of one CNN layer with floating weights.

5.1 Claim 1: Scalability

$\texttt{TTnet}_{16-|8}$ shows an accuracy of 41.6% on ImageNet with truth tables of size $n = 16$. These results are comparable to those achieved by the original BNN paper [Hubara *et al.*, 2016] and XnorNet [Rastegari *et al.*, 2016], see Table 1. Our experimental results confirm that `TTnet` achieves high accuracy on a large dataset, which proves its non-trivial nature. We emphasize that other differentiable logic gate networks did not scale to ImageNet so far: the work done by [Petersen *et al.*, 2022] does not scale to larger datasets than CIFAR-10. However, we believe there remains room for improvement on `TTnet` in terms of accuracy as BNNs now can achieve $\approx 70\% - 75\%$ [Liu *et al.*, 2018] on ImageNet. Detailed architecture/training information are given in Appendix.

Table 2 studies the influence of the input bit size n of the truth table on the scalability for CIFAR-10.

5.2 Claims 2: Compactness

We provide experimental evidence to support our claim of providing easy-to-train, accurate, and compact logic gate DCNNs. The results are presented in Table 3 for tabular datasets (Adult and Breast Cancer) and Table 4 for image datasets (MNIST and CIFAR-10) along with comparisons with state-of-the-art differentiable logic gate DNN [Petersen *et al.*, 2022], BNNs [Umuroglu *et al.*, 2017; Hirtzlin *et al.*, 2019] and SNNs [Molchanov *et al.*, 2017; Mocanu *et al.*, 2018; Han *et al.*, 2016; Zeng and Urtasun, 2018; Zhou *et al.*, 2021]. More information on training conditions, architectures, and additional comparisons can be found in Appendix.

A FLOP (FLoating point OPerations) is typically composed of multiple binary operations (OPs). Float32 adders/multipliers require around 1,000 logic gates or lookup tables and have a significant delay. They are commonly implemented in CPU and GPU hardware due to their importance. However, they are much more expensive than simple calculations or bitwise logical operations on int64 data types. Processors can perform 3 to 10 int64 bitwise operations per cycle, while floating-point operations usually take a full clock cycle. Converting a non-sparse model assumes a conservative estimate of 1000 OPs per 1 FLOP. Neural networks' speeds are theoretical, with sparse execution often being 10 to 100 times slower. In practice, 1,000 binary OPs for 1 clear FLOP (float32) is a conservative estimate for SNNs. It's also a cautious estimate for sparse float32 models. Theoretical estimates assume no density cost and no hardware acceleration for floating-point operations [Petersen *et al.*, 2022].

Performances discussion - small datasets. We present a comparison between two methods, $\texttt{TTnet}_{6-4}$ and Diff Logic Net, with respect to accuracy and compactness on small

Datasets	Adult		Breast Cancer	
	Acc.	OPs	Acc.	OPs
Diff Logic Net	84.8%	1280	76.1%	640
$\texttt{TTnet}_{6-4}$	85.3%	895	77.6%	71

Table 3: Comparison of **TTnet** with state-of-the-art Diff Logic Net [Petersen *et al.*, 2022] on Adult and Breast Cancer tabular datasets for two models. Acc. stands for accuracy, OPs for the number of logical gates.

datasets. Compactness is defined in terms of the number of Boolean logic gates (NAND/NOR/AND/OR/XOR/XNOR) used by the models, noted as OPs. Our results in Table 3 demonstrate that $\texttt{TTnet}_{6-4}$ outperforms Diff Logic Net in terms of accuracy for both tabular datasets, while using much fewer gates. On the MNIST dataset, as indicated in Table 4 (top table), both models achieve comparable accuracies (10% maximum relative error rate), but $\texttt{TTnet}_{6-4}$ utilizes fewer operations. In summary, our findings show that TTnet and Diff Logic Net have similar performance on small datasets, with TTnet offering the advantage of compactness in terms of number of logical gates.

MNIST		Acc.	# Param.	OPs	FLOPs
Traditional models	Linear Regression	91.60%	4K	(4M)	4K
	Neural Network	98.40%	22.6M	(45G)	45M
Boolean DNNs	Diff Logic Net (small)	97.69%	48K	48K	-
	Diff Logic Net	98.47%	384K	384K	-
	$\texttt{TTnet}_{6-4}$ (small)	97.44%	37K	34K	-
	$\texttt{TTnet}_{6-4}$ (big)	98.32%	203K	188K	-
BNNs	FINN	98.40%	-	5.28M	-
SNNs	M17	98.08%	4K	(8M)	8K
	SET-MLP	98.74%	89.8K	(180M)	180K

CIFAR-10		Acc.	# Param.	OPs	FLOPs
Boolean DNNs	Diff Logic Net (small)	51.27 %	48K	48K	-
	Diff Logic Net (medium)	57.39 %	512K	512K	-
	Diff Logic Net (large)	60.78 %	1.28M	1.28M	-
	Diff Logic Net (large x2)	61.41 %	2.56M	2.56M	-
	Diff Logic Net (large x4)	62.14 %	5.12M	5.12M	
	$\texttt{TTnet}_{6-4}$	50.10 %	565K	565K	-
	$\texttt{TTnet}_{12-4}$	70.75 %	189M	189M	-
	$\texttt{TTnet}_{12-\|4}$	84.63 %	1.2G	1.2G	-
BNNs	H19	91.00%	23.9 M	87.4G	-
SNNs	PBW (ResNet32)	38.64 %	-	(140M)	(140K)
	MLPrune (ResNet32)	36.09 %	-	(140M)	(140K)
	ProbMask (ResNet32)	76.87 %	-	(140M)	(140K)
	SET-MLP	74.84 %	279K	(558M)	558K

Table 4: Comparisons on MNIST (top table) and CIFAR-10 (bottom table) of **TTnet** with state-of-the-art Diff Logic Net [Petersen *et al.*, 2022], traditional models, Binary Neural Networks (BNNs: FINN [Umuroglu *et al.*, 2017] and H19 [Hirtzlin *et al.*, 2019]) and Sparse Neural Networks (SNNs: M17 [Molchanov *et al.*, 2017], SET-MLP [Mocanu *et al.*, 2018], PBW [Han *et al.*, 2016], MLPrune [Zeng and Urtasun, 2018] and ProbMask [Zhou *et al.*, 2021]). Estimated operations (OPs or FLOPs) are provided in brackets. Note that for MNIST we did not count the number of gates needed to perform the final layer (360K for $\texttt{TTnet}_{6-4}$ (small) and 2.9M for $\texttt{TTnet}_{6-4}$ (big)), as in [Petersen *et al.*, 2022].

Performances discussion - large dataset. We compare again $\texttt{TTnet}_{6-4}$ and Diff Logic Net, but on the CIFAR-10 dataset, specifically focusing on scalability in terms of accuracy, training time, and model compactness. As shown in Table 4 (bottom table), for the small model, Diff Logic Net exhibits higher compactness with comparable accuracy than TTnet. On the other hand, when considering larger models, TTnet achieves an accuracy of 70.75% while Diff Logic Net shows a maximum of 62.14%. The superior scalability of TTnet in terms of accuracy is achieved at the cost of a more complex Boolean logic circuit.

To investigate the impact of relaxing the compactness constraint, we demonstrate that $\texttt{TTnet}_{12-|4}$ can achieve an accuracy of 84.63% with 1.2G OPs. As the dataset gets larger and more complex, the complexity of the Boolean circuit needs to increase as well to maintain high accuracy. This is achievable with TTnet due to its scaling training properties.

Indeed, TTnet is notably easier to train than Diff Logic Net: $\texttt{TTnet}_{6-4}$ achieves an accuracy of 70% in under 1 hour of training and $\texttt{TTnet}_{12-|4}$ reaches 84.63% after 20 hours (on an NVIDIA GeForce RTX 3090 GPU), while the Diff Logic Net requires 90 hours of training (on a single NVIDIA A6000 GPU) to reach a 62.14% accuracy. We note these two GPUs are equivalent in terms of image training time, see Supplementary Material. The accuracy achieved by TTnet on CIFAR-10, along with its effective training, underscores its potential as a differentiable CNN logic gate architecture.

5.3 Claim 3: Complete, Sound Formal Verification

Greater Boolean logic compactness in circuit design can aid formal verification by reducing the complexity of the verification process. Therefore, we applied TTnet to formal verification, where two strategies are common: either using a specific solver to verify ReLU based-DNN like β-Crown [Wang *et al.*, 2021], or using a specific architecture that allows a generic verification method. We compared both methods based on natural accuracy, verified accuracy for l_∞-norm bounded input perturbations (formal definition in Appendix.For a fair comparison, we note that our pre-processing layer and training configuration are the same as [Jia and Rinard, 2020]. More results and discussions can be found in Appendixee.

	General DNN + α-β-Crown [Xu *et al.*, 2020] [Wang *et al.*, 2021]		$\texttt{TTnet}_{9-1}$ + General SAT verification pipeline	
	Verif. time (s)	Timeout (%)	Verif. time (s)	Timeout (%)
MNIST	96	13	0.06 ($\times$1600)	0
CIFAR-10	175	27	0.14 ($\times$1250)	0

Table 5: Comparison of verification strategies: usage of a general DNN to verify with $\alpha-\beta$-Crown [Xu *et al.*, 2020; Wang *et al.*, 2021] or using specific TTnet with a general SAT verification method. The comparison is based on the 7 benchmarks from the VNN competition, and the results are presented as an average, full results are given in Appendix. TTnet has no pre-processing, $n = 9$, $k = 1$.

Strategy 1: Comparison with general DNN solvers. In Table 5, we present a comparison of our proposed verification strategy, which utilizes the TTnet architecture and classical verification tools, against the state-of-the-art $\alpha - \beta$-Crown method, the winner of the VNN competition 2021. The comparison is based on the 7 benchmarks from the VNN competition, and the results are presented as an average. Our approach demonstrates a significant improvement in verification time, with an average speed-up of 1250x for CIFAR-10 and 1600x for MNIST, at the same noise level. Additionally, a higher verified accuracy (+4% and +7%) was observed on the cifar_10_resnet benchmark in Appendix.Also, our approach did not encounter any timeouts, whereas $\alpha - \beta$ Crown had an average of more than 10% of timeouts.

It should be noted that our strategy cannot be directly compared to the VNN competition as the competition focuses on novel DNN verification algorithms/pipelines, whereas we propose a new DNN family (`TTnet`) that can be easily verified using classical verification tools. However, the results presented demonstrate the competitiveness of our approach. One can use our strategy to verify the robustness property on CIFAR-10, 1K images, standard DNN with $\alpha - \beta$-Crown which takes 2 days, whereas it takes 14 seconds if one chooses to verify `TTnet`.

Dataset (noise)	**Complete method**	**Accuracy** Verif.	Nat.	**Verif. time** (s)	**Timeout**
MNIST (ϵ_{test} = 0.1)	$\texttt{TTnet}_{9-1}$	**95.12%**	98.33%	**0.012**	0
	JR20	91.68%	97.46%	0.1115	0
	N+19 *	20.00%	96.00 %	5	0
MNIST (ϵ_{test} = 0.3)	$\texttt{TTnet}_{9-1}$	66.24%	97.43 %	**0.065**	0
	JR20	**77.59%**	96.36%	0.1179	0
CIFAR-10 (ϵ_{test} = 2/255)	$\texttt{TTnet}_{9-1}$	**32.32%**	49.23%	**0.06**	0
	JR20	30.49%	47.35%	0.1750	0
CIFAR-10 (ϵ_{test} = 8/255)	$\texttt{TTnet}_{9-1}$	21.08%	31.13%	**0.04**	0
	JR20	**22.55%**	35.00%	0.1781	0

* *results given on the first 1K images of the test set. Moreover, the authors only authorize a maximum of 20 pixels to switch.*

Table 6: Application of `TTnet` to complete adversarial robustness verification for low and high noise bounded by l_∞. We tabulate results of verified accuracy, natural accuracy, and mean verification time on MNIST and CIFAR-10 datasets in comparison to state-of-the-art SAT methods (JR20 stands for [Jia and Rinard, 2020] and N+19 for [Narodytska *et al.*, 2019]). The best verified accuracy and verification time are displayed in bold.

Strategy 2: comparison with BNNs. We compared our work with the state-of-the-art of exact verification for BNNs [Jia and Rinard, 2020; Narodytska *et al.*, 2019]. As shown in Table 6, our verified accuracy is competitive to the one of BNNs in all cases. In low noise case, we slightly outperform BNNs [Jia and Rinard, 2020] in terms of verified accuracy and verification time. For high noise, we offer a trade-off: a slightly better verification time for a slightly lower verified accuracy. In addition, we experienced a better resolution time than BNNs while using a general SAT solver, namely MiniCard [Liffiton and Maglalang, 2012]. We highlight that in [Jia and Rinard, 2020] the SAT solver is custom-made and specific to their problem, while in our case we can use a general one. We tested 9 SAT solvers [Ignatiev *et al.*, 2018] on MNIST high noise: the best one is MiniCard (0.0008s), the worst is MapleCM (0.0033s); while MiniSat takes 0.242s for BNNs in [Jia and Rinard, 2020].

6 Limitations And Future Works

Overall limitations. The main limitation of `TTnet` is closely related to the maximum input bit size n, so that the truth table can be fully enumerated practically. Additionally, users cannot pre-set the final number of gates or complexity before starting the training process. Also, as mentioned in Section 2, it has not yet been shown that the aggregation of local features learned by non-linear functions can provide good performances for another dataset than images and tabular.

Limitations on scaling. While `TTnet` has shown promising results, its performance falls behind state-of-the-art computer vision models on larger datasets. Improving the natural accuracy of the model without increasing n is an area for future investigation [Liu *et al.*, 2018; Bello *et al.*, 2021].

Limitations on compactness. `TTnet` achieves better compactness than previous methods on tabular and MNIST datasets, but its compactness on CIFAR-10 and accuracy on MNIST could be improved. Further research is needed to address this limitation and explore the use of CNNs with Boolean logic circuits for larger datasets such as ImageNet.

Limitations on formal verification. While `TTnet` has demonstrated promising results in terms of verification, it may be less effective in the presence of high noise. Further research is needed to increase the model's verified accuracy, especially in noisy environments. This includes exploring the use of dedicated robustness training techniques.

Future works. The `TTnet` architecture presents several opportunities for future research. These include exploring its application on time series or graph-based datasets, developing better LTT architectures and training methods to achieve higher accuracy on ImageNet, and creating heuristics for converting truth tables to optimized CNF for larger n, which would in turn help the scaling. `TTnet` could also be a good candidate for the VNN competition.

7 Conclusion

This work is a step towards more scalable, compact, and verifiable DNNs while linking symbolic AI and learning AI. There is room for improving `TTnet`, and we hope that it will inspire further explorations into the use of truth tables as a tool for applying DNNs to critical applications. We provide more discussions on `TTnet` in Appendix.

References

[Agarwal *et al.*, 2021] Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E Hinton. Neural additive models: Interpretable machine learning with neural nets. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[AI, 2023] NIST AI. Artificial Intelligence Risk Management Framework (AI RMF 1.0). 2023.

[Arora and Barak, 2009] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[Bello *et al.*, 2021] Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[Blake, 1938] Archie Blake. Corrections to Canonical expressions in Boolean algebra. *Journal of Symbolic Logic*, 1938.

[Brendel and Bethge, 2018] Wieland Brendel and Matthias Bethge. Approximating CNNs with Bag-of-local-Features models works surprisingly well on ImageNet. In *International Conference on Learning Representations (ICLR)*, 2018.

[Brix *et al.*, 2023] Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T Johnson, and Changliu Liu. First three years of the international verification of neural networks competition (VNN-COMP). *International Journal on Software Tools for Technology Transfer*, 2023.

[Chatterjee, 2018] Satrajit Chatterjee. Learning and memorization. In *International conference on machine learning (ICML)*, 2018.

[Commission, 2021] European Commission. Proposal for a regulation laying down harmonised rules on artificial intelligence. *Official Journal of the European Union*, 2021.

[Daemen and Rijmen, 2002] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[Driscoll, 2020] Michael Driscoll. System and method for adapting a neural network model on a hardware platform, 2020. US Patent App. 16/728,884.

[Goodfellow *et al.*, 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[Han *et al.*, 2016] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

[Hirtzlin *et al.*, 2019] Tifenn Hirtzlin, Bogdan Penkovsky, Marc Bocquet, Jacques-Olivier Klein, Jean-Michel Portal, and Damien Querlioz. Stochastic computing for hardware implementation of binarized neural networks. *IEEE Access*, 2019.

[Hubara *et al.*, 2016] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[Ignatiev *et al.*, 2018] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2018.

[Jia and Rinard, 2020] Kai Jia and Martin Rinard. Efficient exact verification of binarized neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[Jia and Rinard, 2021] Kai Jia and Martin Rinard. Exploiting verified neural networks via floating point numerical error. In *International Static Analysis Symposium*, 2021.

[Klir *et al.*, 1997] George J Klir, Ute St. Clair, and Bo Yuan. *Fuzzy set theory: foundations and applications*. Prentice-Hall, Inc, 1997.

[Liffiton and Maglalang, 2012] Mark H Liffiton and Jordyn C Maglalang. A cardinality solver: More expressive constraints for free. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2012.

[Liu *et al.*, 2018] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[Mocanu *et al.*, 2018] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 2018.

[Molchanov *et al.*, 2017] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International conference on machine learning (ICML)*, 2017.

[Müller *et al.*, 2022] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. PRIMA: general and precise neural network certification via scalable convex hull approximations. *Proceedings of the ACM on Programming Languages*, 2022.

[Narodytska *et al.*, 2019] Nina Narodytska, Hongce Zhang, Aarti Gupta, and Toby Walsh. In search for a SAT-friendly binarized neural network architecture. In *International Conference on Learning Representations (ICLR)*, 2019.

[Paszke *et al.*, 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[Petersen *et al.*, 2022] Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Deep differentiable logic gate networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[Rastegari *et al.*, 2016] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[Roussel and Manquinho, 2009] Olivier Roussel and Vasco Manquinho. Pseudo-Boolean and cardinality constraints. In *Handbook of satisfiability*. IOS Press, 2009.

[Rumelhart *et al.*, 1986] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 1986.

[Udovenko, 2023] Aleksei Udovenko. DenseQMC: an efficient bit-slice implementation of the Quine-McCluskey algorithm. *arXiv*, 2023.

[Umuroglu *et al.*, 2017] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the ACM/SIGDA International Symposium On Field-Programmable Gate Arrays*, 2017.

[Van Der Walt *et al.*, 2011] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 2011.

[Wang *et al.*, 2019] Erwei Wang, James J Davis, Peter YK Cheung, and George A Constantinides. LUTNet: Rethinking inference in FPGA soft logic. In *Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019.

[Wang *et al.*, 2021] 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 neural network robustness verification. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[Xu *et al.*, 2020] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers. In *International Conference on Learning Representations (ICLR)*, 2020.

[Zeng and Urtasun, 2018] Wenyuan Zeng and Raquel Urtasun. Mlprune: Multi-layer pruning for automated neural network compression. *openreview*, 2018.

[Zhou *et al.*, 2021] Xiao Zhou, Weizhong Zhang, Hang Xu, and Tong Zhang. Effective sparsification of neural networks with global sparsity constraint. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.