FORMAL EXPLANATIONS OF LOGIC GATE NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Logic Gate Networks (LGNs) are a Boolean alternative to conventional neural networks (NNs) offering competitive accuracy while enabling highly efficient inference. While their Boolean circuit nature is attractive from an implementation perspective, it also raises a natural question: are LGNs more amenable to formal reasoning and explainability than conventional NNs? In this work, we answer this question through the computation of formal abductive explanations (AXps). We develop a propositional encoding of LGNs that captures their complete inference mechanism, including standard hidden layers, convolutional layers, and the voting mechanism. This enables us to test feature relevance underlying any AXp computation by querying a SAT solver. Since encoding the voting mechanism requires potentially expensive cardinality constraints, we additionally develop a MaxSAT-based reasoning approach that avoids their explicit CNF representation. Experiments across a range of datasets demonstrate that formal explanations of LGNs can be computed several orders of magnitude faster than explanations of conventional NNs using a dedicated state-of-the-art formal explainer. Moreover, we observe that LGNs tend to admit substantially shorter explanations. Hence, our results suggest that, beyond efficient inference, the inherently Boolean structure of LGNs offers significant advantages for formal explainability.

1 INTRODUCTION

Deep Differentiable Logic Gate Networks (LGNs) (Petersen et al., 2022) are neural networks whose discretized computational layers comprise regular Boolean logic gates, such as AND and XOR, acting on Boolean signals. Their key innovation is learning a distribution over all 16 possible two-input Boolean functions at each gate, using continuous relaxations to enable gradient-based training. They have proven to be scalable and accurate alternatives to conventional neural networks. Recent work includes convolutional and pooling layers (Petersen et al., 2024), learnable connections (Fojcik et al., 2026), and more efficient representations of gate functions during training (Gerlach et al., 2026b). A key advantage of LGNs is that their Boolean computation supports very fast inference.

The rapid rise of artificial intelligence in decision making (LeCun et al., 2015; Jordan & Mitchell, 2015; Mnih et al., 2015; ACM, 2018) has led to a demand for eXplainable Artificial Intelligence (XAI) (Miller, 2019; Guidotti et al., 2019). Rather than simply accepting the decisions made by black-box AI systems, there is now considerable demand for an AI system to be able to *explain* its decisions to a human (Ribeiro et al., 2016; Lundberg & Lee, 2017). Many explanation methods, however, lack formal guarantees (Weber et al., 2024) (Marques-Silva & Huang, 2024). One well-developed approach, *formal explainable AI* (Shih et al., 2018; Marques-Silva & Ignatiev, 2022), builds on formal reasoning about the models of interest. It computes explanations that capture the semantics of the model and answer why a particular decision was made, or why another decision was not made. The computational cost of this reasoning remains a central challenge.

Recent work has already demonstrated the advantages of Boolean network representations for formal verification. For example, binarized neural networks have been studied in this context (Hubara et al., 2016; Narodytska et al., 2018), while Truth Table Nets (Benamira et al., 2024) aim to make neural network verification more tractable. A SAT encoding of LGNs has also been used to check global robustness and fairness (Kresse et al., 2025). Our focus is instead on explaining individual predictions, including those of LGNs with convolutional and pooling layers. Specifically, we seek subset-minimal sets of feature values sufficient to guarantee a prediction. For conventional neural networks, existing methods (Ignatiev et al., 2019; Izza et al., 2024; Wu et al., 2026) compute such

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

explanations over the full input domain or within a specified perturbation region. These methods provide a basis for comparing formal explainability across the two network architectures.

In this paper, we argue that the advantages of LGNs extend beyond fast inference: their direct Boolean encodings make them particularly amenable to SAT-based formal explanation (Biere et al., 2021). We develop SAT- and MaxSAT-based approaches for computing abductive explanations of LGNs, including networks with convolutional and pooling layers. The SAT formulation encodes pairwise vote comparisons as cardinality constraints, whereas the MaxSAT formulation models the voting mechanism as an objective function, following the ideas of (Ignatiev et al., 2022). We further reduce the encodings through network simplification, including gate equivalence and constant detection under feature-domain constraints.

Our experiments evaluate formal explanation extraction on tabular and image datasets and compare LGNs with conventional ReLU networks, using VeriX+ (Wu et al., 2026) for the latter. The results show that, for the models considered, LGNs can be explained substantially more efficiently than ReLU networks of comparable size or predictive accuracy. Thus, the advantages of Boolean computation extend beyond fast inference to efficient formal explainability.

2 PRELIMINARIES

2.1 CLASSIFICATION PROBLEMS

We assume classification problems classify data instances into classes $\mathcal{K}$ where $|\mathcal{K}| = k \geq 2$. We are given a set of m features $\mathcal{F}$, where the value of feature $i \in \mathcal{F}$ is in a domain $\mathbb{D}_i$, which may be Boolean, (bounded) integer or real. The *complete feature space* is defined by $\mathbb{F} \triangleq \prod_{i=1}^{m} \mathbb{D}_i$.

A *data point* in feature space is denoted $\mathbf{v} = (v_1, \ldots, v_m)$ where $v_i \in \mathbb{D}_i, 1 \leq i \leq m$. An *instance* of the classification problem is a pair of feature vector and its corresponding class, i.e. $(\mathbf{v}, c)$, where $\mathbf{v} \in \mathbb{F}$ and $c \in \mathcal{K}$. We use the notation $\mathbf{x} = (x_1, \ldots, x_m)$ to represent an arbitrary point in feature space, where each x_i will take a value from $\mathbb{D}_i$.

A *classifier* is a total function from feature space to class: $\kappa : \mathbb{F} \to \mathcal{K}$. Many approaches exist to define classifiers including decision sets (Clark & Boswell, 1991; Lakkaraju et al., 2016), decision lists (Rivest, 1987), decision trees (Hyafil & Rivest, 1976), random forests (Friedman, 2001), boosted trees (Chen & Guestrin, 2016), neural nets (Nair & Hinton, 2010; Hubara et al., 2016), and logic gate networks (Petersen et al., 2022).

2.2 LOGIC GATE NETWORKS

Logic gate networks (LGNs) (Petersen et al., 2022) are a Boolean neural net approach that uses standard Boolean gates to build the network, as opposed to binarized neural networks (Hubara et al., 2016) that use Pseudo-Boolean constraint gates. Generally, an LGN implements a layered Boolean circuit whose computational nodes implement two-input Boolean functions.

For a (binary) point $\mathbf{x} \in \mathbb{F}$, we denote the Boolean value at node v of the LGN evaluated on input $\mathbf{x}$ by $z(\mathbf{x})$. Each node v of the LGN receives values from two fixed predecessors u_1, u_2 and computes

$$z_v(\mathbf{x}) = g_v\big(z_{u_1}(\mathbf{x}), z_{u_2}(\mathbf{x})\big),$$

where g_v is selected from the set of all 16 two-input Boolean functions, including constants, projections, and binary logical operators. Importantly, unlike conventional neural networks, LGN training is parameterized by *gate choices* rather than connection weights. The final-layer gates are partitioned into class-specific groups $O_1, \ldots, O_C$, yielding scores and voting-based predictions

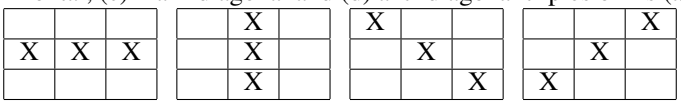
$$s_c(\mathbf{x}) = \sum_{v \in O_c} z_v(\mathbf{x}), \qquad \widehat{y}(\mathbf{x}) = \operatorname*{arg\,max}_{c \in \{1,\ldots,C\}} s_c(\mathbf{x}),$$

with a fixed tie-breaking rule.

After training, each node is assigned its highest-probability gate, thus producing a final LGN. While the original architecture (Petersen et al., 2022) makes use of only *dense layers* above and randomly initialized gate connectivity, (Petersen et al., 2024) applies residual initialisation (He et al., 2016) and adds support for convolutional layers and OR-pooling.

Example 1: Consider a problem of a 4×4 tic-tac-toe. X tries to put three Xs in a row, orthogonally or diagonally, anywhere on the board. O tries to put three Os in a row. There are 4×4 categorical input features { X, O, empty }. The output is **O** if O generates more triples, and **✖** otherwise.

We can construct an LGN for this problem having 1 convolutional layer with 8 kernels looking for (a) vertical, (b) horizontal, (c) main-diagonal and (d) anti-diagonal triples of Xs (and also Os):



(a) (b) (c) (d)

We then build an output layer that compares the number of X patterns found in the grid, versus the number of O patterns, and outputs **O** if the number of O patterns is greater, otherwise outputting **✖**. This LGN defines a classifier κ from 16 features (the 4x4 grid) the class set $\mathcal{K} = \{\text{✖}, \text{O}\}$. □

2.3 Satisfiability and Minimal Unsatisfiability

We assume standard definitions for propositional satisfiability (SAT) and maximum satisfiability (MaxSAT) solving (Biere et al., 2021). A propositional formula is said to be in *conjunctive normal form* (CNF) if it is a conjunction of clauses. A *clause* is a disjunction of literals. A *literal* is either a Boolean variable or its negation. Whenever convenient, clauses are treated as sets of literals while CNF formulas are treated as *sets of clauses*. A truth assignment maps each variable of a formula to a value from $\{0, 1\}$. By definition, a literal $\neg x$ is assigned value 1 iff variable x is assigned value 0. Given a truth assignment, a clause is said to be satisfied if at least one of its literals is assigned value 1; otherwise, it is falsified by the assignment. A formula ϕ is satisfied if all of its clauses are satisfied; otherwise, it is falsified. If there exists no assignment that satisfies a CNF formula, then the formula is *unsatisfiable*, which is often written as $\phi \models \bot$.

In the context of unsatisfiable formulas, the maximum satisfiability (MaxSAT) problem is often of interest (Biere et al., 2021, Chapters 23 and 24). In partial unweighted MaxSAT, a formula ϕ is a conjunction of *hard* clauses $\mathcal{H}$, which must be satisfied, and *soft* clauses $\mathcal{S}$, which represent a preference to satisfy those clauses, i.e. $\phi = \mathcal{H} \land \mathcal{S}$ (or $\phi = \mathcal{H} \cup \mathcal{S}$ in the set theory notation). So the MaxSAT problem consists in finding an assignment that satisfies all the hard clauses and maximizes the total number of satisfied soft clauses. Equivalently, the MaxSAT problem is often formulated as maximizing a linear objective function *defined* by $\mathcal{S}$ subject to satisfying the constraints $\mathcal{H}$.

In the analysis of an unsatisfiable formula ϕ, one is also often interested in identifying minimal unsatisfiable subsets (MUSes) of ϕ (Biere et al., 2021, Chapter 14). Given a formula $\phi = \mathcal{H} \land \mathcal{S} \models \bot$, a subset of clauses $\mu \subseteq \mathcal{S}$ is an MUS iff $\mathcal{H} \land \mu \models \bot$ and $\forall \mu' \subsetneq \mu$ it holds that $\mathcal{H} \land \mu' \not\models \bot$. Informally, an MUS can be seen as a minimal explanation of unsatisfiability of formula ϕ as it provides the minimal information that needs to be added to the hard clauses $\mathcal{H}$ to obtain unsatisfiability.

2.4 Formal Abductive Explanations

We consider a typical classification scenario: given a data point $\mathbf{v}$, classifier κ classifies it as class $\kappa(\mathbf{v})$. A *post-hoc explanation* of the prediction of κ on data point $\mathbf{v}$ seeks to explain $\kappa(\mathbf{v})$.

An *abductive explanation* (AXp) is a minimal set of features $\mathcal{X}$ such that any data point sharing the same feature values with $\mathbf{v}$ on these features is guaranteed to be assigned the same class $c = \kappa(\mathbf{v})$ (Shih et al., 2018; Ignatiev et al., 2019). Formally, $\mathcal{X}$ is a subset-minimal set of features s.t.

$$\forall(\mathbf{x} \in \mathbb{F}).\ \Big[\bigwedge\nolimits_{i \in \mathcal{X}} (x_i = v_i)\Big] \rightarrow (\kappa(\mathbf{x}) = c) \tag{1}$$

Some literature (Shih et al., 2018; Darwiche & Hirth, 2020) refers to abductive explanations as prime implicant (PI) explanations or sufficient reasons. It is known (Ignatiev et al., 2020) that formal abductive explanations for ML predictions are related with the concept of MUSes (defined earlier) of an *unsatisfiable* formula encoding the ML classification process $\kappa(\mathbf{v}) = c$, namely if one represents $[\kappa(\mathbf{x}) \neq c]$ as hard clauses and $[\bigwedge_{i=1}^{m} (x_i = v_i)]$ as soft clauses.

While a dual concept of *contrastive explanations* (CXps) exist (and our prototype supports them) (Miller, 2019; Marques-Silva & Ignatiev, 2022), our work focuses solely on AXp computation, which is motivated by the limitations of the experimental competition used in Section 4.

3 MAKING LGNS AMENABLE TO FORMAL REASONING

Although LGNs largely reduce to combinational circuits, their logical encoding is not trivial as naive approaches face challenges including the voting mechanism and input feature domains. This section overviews our approach to encode all the components of a trained logic gate network into CNF.

3.1 FEATURE BINARIZATION

LGNs take varied features as input, routed through a binarization layer to interface with discrete logic gates. Here, categorical features are binarized using the standard one-hot encoding (OHE) (Pedregosa et al., 2011) while ordinal / numeric features are binarized with the use of thermometer encoding (Buckman et al., 2018). In either of these cases, to ensure correct formal reasoning about the possible feature values, we need to express logically the domains of all non-binary input features.

Given a categorical feature $i \in \mathcal{F}$ with the domain $\mathbb{D}_i$, the OHE produces $|\mathbb{D}_i|$ binary features x_{ij}, each representing a value $j \in \mathbb{D}_i$, with the constraint that exactly one of them can take value 1 at a time. This can be done by adding a CNF representation of the cardinality constraint $\sum_{j \in \mathbb{D}_i} x_{ij} = 1$. There are various options to encode cardinality constraints into CNF (Biere et al., 2021, Chapter 28).

Example 2: Consider the LGN defined in Example 1. Each cell (i, j) of the grid can have one of 3 possible values: X, O, or empty. Thus, the OHE introduces 3 Boolean variables. For consistency with the other examples, let us denote them as: x_{ij}, o_{ij}, and e_{ij}. To ensure exactly one value is used for cell (i, j) at a time, a CNF encoding of $x_{ij} + o_{ij} + e_{ij} = 1$ is added. □

For an ordinal feature $i \in \mathcal{F}$ with the domain $\mathbb{D}_i$, the thermometer encoding considers $1 \leq k \leq |\mathbb{D}_i| - 1$ threshold values $t_{i1} < t_{i2} < \ldots < t_{ik}$, $t_{ij} \in \mathbb{D}_i$, and considers k Boolean features x_{ij} with the meaning $x_{ij} = 1$ if and only if $f_i \geq t_{ij}$. In order to encode such a domain $\mathbb{D}_i$, one can use a chain of $k-1$ implications $x_{ij+1} \rightarrow x_{ij}$, which can be naturally represented as clauses $\neg x_{ij+1} \vee x_{ij}$.

Example 3: Consider a continuous feature $f_i \in [-30, 30]$, to be thermometer encoded to a 5-bit representation, so $k = 5$. We can define the 5 thresholds -20, -10, 0, 10, and 20 and introduce 5 Boolean variables x_{ij} such that $x_{ij} = 1 \Leftrightarrow f_i \geq t_{ij}$, and add clauses $(\neg x_{ij+1} \vee x_{ij})$, $1 \leq j \leq 4$. For example, if $f_i = 3.14$, then the 5-bit representation of this value is $(1, 1, 1, 0, 0)$, i.e. $x_{ij} = 1$ for $1 \leq j \leq 3$ and $x_{ij} = 0$ for $j \geq 4$. Similarly, $f_i = -13$ is encoded as a vector $(1, 0, 0, 0, 0)$. □

3.2 DENSE LAYERS

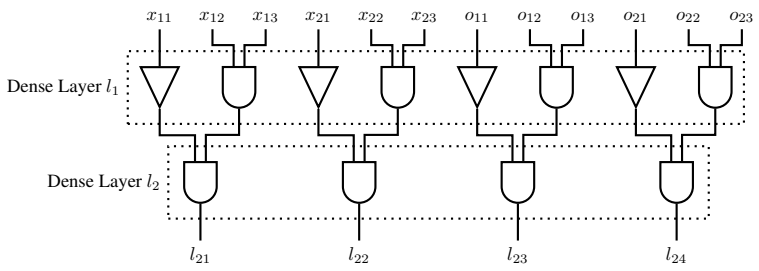
Each dense layer consists of one- and two-input Boolean gates. We encode these gates using the standard Tseitin transformation (Tseitin, 1968), introducing a fresh propositional variable for each gate output and clauses defining its value in terms of the gate inputs. The gates' inputs are represented by Boolean variables from the encoding of the preceding layer. The input layer of the LGN is represented by the Boolean variables encoding the input instance.

Specifically, consider a gate g with input variables u and v and Boolean operation f_g. We introduce an output variable z_g and encode $z_g \leftrightarrow f_g(u, v)$ in CNF. For example, an AND gate is encoded by $(\neg u \vee \neg v \vee z_g) \wedge (u \vee \neg z_g) \wedge (v \vee \neg z_g)$. The other gate operations are encoded analogously. In general, any two-input gate is encoded by one auxiliary variable, using at most four clauses. The encoding of a dense layer is the conjunction of its gate encodings, and the resulting CNF formula is linear in the number of gates. For every assignment to the input features, the gate clauses uniquely determine all intermediate outputs according to the LGN's computation.

Example 4: Consider the LGN of Example 1, for simplicity restricted to a 3×2 game board, so that only horizontal three-in-a-rows are possible, and a particular input:

O		O
X	X	X

Let x_{ij} and o_{ij} denote the binarized input features indicating that the cell in row i and column j contains X or O, respectively.



Observe that the above two dense layers suffice to compute the following functions:

$$l_{21} \leftrightarrow (x_{11} \wedge (x_{12} \wedge x_{13})), \quad l_{22} \leftrightarrow (x_{21} \wedge (x_{22} \wedge x_{23})),$$
$$l_{23} \leftrightarrow (o_{11} \wedge (o_{12} \wedge o_{13})), \quad l_{24} \leftrightarrow (o_{21} \wedge (o_{22} \wedge o_{23})).$$

To illustrate the gate-by-gate encoding, consider the computation of l_{21}. In the first layer, an AND gate combines x_{12} and x_{13}, while a buffer gate copies x_{11}. Introducing intermediate variables t and p, respectively, gives $t \leftrightarrow (x_{12} \wedge x_{13})$ and $p \leftrightarrow x_{11}$. In the second layer, an AND gate combines these intermediate variables as $l_{21} \leftrightarrow (t \wedge p)$. Encoding each such equivalence yields a few clauses, to be added to the CNF. The rest of functions l_{ij} are encoded analogously, using separate intermediate variables. For the board shown above, the binarized input assignment and the clauses produced imply $(l_{21}, l_{22}, l_{23}, l_{24}) = (0, 1, 0, 0)$. These output variables are then used to encode the next layer, or the voting mechanism described in Section 3.4 if this is the final dense layer. □

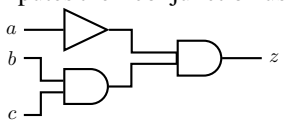
3.3 Convolutional Layers

Motivated by the success of CNNs, (Petersen et al., 2024) extended LGNs with convolutional layers, whose kernels are small binary trees of logic gates. Each kernel is applied at multiple spatial positions, using the same gate operations and relative input connections at every position.

We encode each kernel application using the gate-by-gate Tseitin transformation described in Section 3.2. At each position, the kernel inputs are connected to the variables representing the corresponding outputs of the preceding layer. We introduce fresh variables for the gate outputs and add the clauses defining them. Kernel inputs falling outside the preceding layer's output grid are represented by the constant false, in accordance with zero padding.

Therefore, a kernel applied at n positions requires n copies of its encoding, with different output variables and the appropriate input variables in each copy. The conjunction of these copies, over all kernels, encodes the convolutional layer. Each kernel application produces one output variable. These variables serve as inputs when encoding the next layer. For a kernel containing g gates and applied at n positions, the CNF encoding introduces ng variables and $O(ng)$ clauses.

Example 5: Consider a 4×4 game board and a kernel that detects three horizontally consecutive X symbols. At any position, let a, b, and c be the Boolean variables indicating whether the three selected cells contain X. The kernel computes their conjunction using a buffer gate and two AND gates:



The process can be encoded using the equivalences $p \leftrightarrow a$, $t \leftrightarrow (b \wedge c)$, and $z \leftrightarrow (p \wedge t)$. We translate them into CNF exactly as in Example 4. At the next kernel's position, we repeat the construction with fresh variables p, t, and z and the input variables for the newly selected cells.

With a 3×3 window, stride one, and one-cell zero padding, the kernel has 16 positions and produces a 4×4 feature map. Therefore, its encoding contains 16 copies of the kernel's constraints. Using eight kernels (one for each combination of two symbols and four possible directions in the grid) results in 128 such copies and 128 output variables, in addition to the intermediate variables. □

3.4 VOTING MECHANISM

The final voting layer of an LGN is encoded through pairwise comparisons between the original predicted class $c \in \mathcal{K}$ and each alternative class $d \in \mathcal{K}$. Let $\delta_{cd} = 0$ if the tie-breaking rule favours class d over c, and $\delta_{cd} = 1$ otherwise. For lexicographic tie-breaking, $\delta_{cd} = 0$ when $d < c$ and $\delta_{cd} = 1$ when $c < d$. Then d defeats c precisely when $s_d(\mathbf{x}) - s_c(\mathbf{x}) \geq \delta_{cd}$.

Let ϕ denote the CNF encoding of the preceding layers and input-domain constraints discussed above. To check whether a feature set $\mathcal{X}$ guarantees prediction c, i.e. that (1) holds, we query each pair (c, d) fixing the features in $\mathcal{X}$ to their values given in the instance being explained. A counterexample to $\mathcal{X}$ exists if *any* alternative class can defeat c. (Note that such a class does not need itself be the overall winner.) Our encoding checks the existence of such a counterexample.

Voting with SAT. Using the final-layer variables z_v of the LGN, we express each pairwise comparison of classes c and d as one cardinality constraint:

$$\sum_{v \in O_d} z_v + \sum_{v \in O_c} \neg z_v \geq |O_c| + \delta_{cd}.$$

Note that this constraint is equivalent to the score comparison between classes c and d because $\sum_{v \in O_d} z_v + \sum_{v \in O_c} \neg z_v = s_d(\mathbf{x}) - s_c(\mathbf{x}) + |O_c|$. For each alternative class d, we encode the cardinality constraint into CNF (Biere et al., 2021, Chapter 28) and conjoin it with ϕ. Satisfiability under the conjunction $\bigwedge_{i \in \mathcal{X}} (x_i = v_i)$ produces a counterexample to prediction $\mathcal{X}$. However, if all pairwise queries are unsatisfiable, then $\mathcal{X}$ guarantees the prediction c.

Voting in MaxSAT. Following (Ignatiev et al., 2022), we can avoid explicitly encoding the score comparison by placing the votes in the MaxSAT objective subject to formula ϕ. For each pair (c, d), we use ϕ as hard clauses and introduce the unit-weight soft clauses $S_{cd} = \{(z_v) \mid v \in O_d\} \cup \{(\neg z_v) \mid v \in O_c\}$, which effectively "emulate" the maximization of the objective function $s_d(\mathbf{x}) - s_c(\mathbf{x})$. Class d can defeat c if and only if the maximum value of the objective function is greater than δ_{cd} (alternatively, if the maximum number of satisfied clauses in S_{cd} is at least $|O_c| + \delta_{cd}$). If the optimum is below this bound for every alternative class, then feature set $\mathcal{X}$ guarantees prediction c.

3.5 REDUCING ENCODING SIZE

AXp extraction typically involves repeated calls to a formal reasoning engine to test whether selected feature values suffice to preserve the prediction. Reducing the encoding size can improve the efficiency of these calls. We therefore simplify both the LGN and its CNF encoding before explanation extraction.

On the LGN side, established techniques include merging gates that apply the same Boolean operation to the same inputs, propagating constants, and bypassing identity gates. These are offered by the Torchlogix toolkit (Gerlach et al., 2026a) we use. Additionally, we apply constrained LGN simplification discussed next followed by CNF simplifications. For the latter, we use (1) PySAT's recursive elimination of constants during the encoding process (Ignatiev et al., 2024), and (2) two rounds of CNF formula preprocessing techniques (Biere et al., 2021, Chapter 9). Note that CNF preprocessing is set to preserve the input and voting variables.

Constrained Network Simplification. While tools like Torchlogix are capable of merging gates, removing redundant gates, as well as detecting constant gates, one can also apply formal reasoning subject to feature domain constraints Section 3.1 for checking if further network simplifications are possible. As an example, assume a categorical feature $k \in \mathcal{F}$ has 3 values, which requires three Boolean variables x_{k1}, x_{k2}, and x_{k3}. The OHE of feature k enforces that $x_{k1} + x_{k2} + x_{k3} = 1$. Now, imagine that among the many gates in our LGN there are two distinct gates $g_i = x_{k1} \vee x_{k2}$ and $g_j = x_{k1} \oplus x_{k2}$. These gates are not equivalent over unrestricted Boolean inputs, but become equivalent under the feature-domain constraints. Concretely, as we know the relationship between variables x_{k1} and x_{k2} in all *valid input instances*, gates g_i and g_j turn out to output exactly the same value for any such instance. As a result, one of the gates can be declared a duplicate of the other and removed.

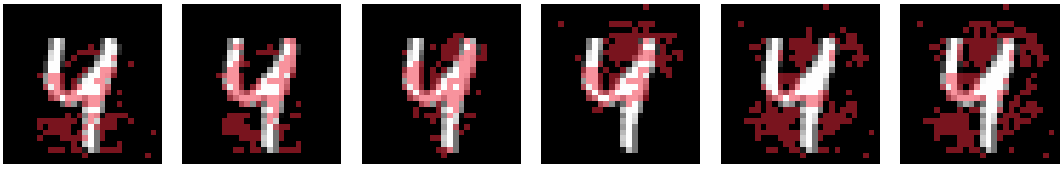
Importantly, LGNs can exhibit gate redundancies subject to feature domain constraints at various levels, i.e. not necessarily for the gates directly connected to the inputs. In general, let ϕ be the CNF encoding of the network's gate definitions and let ψ encode the feature-domain constraints. For gates g_i and g_j with output variables z_i and z_j, let τ_{ij} be the CNF encoding of $t_{ij} \leftrightarrow (z_i \oplus z_j)$. With this, the gates are equivalent subject to domain constraints iff formula $\phi \land \psi \land \tau_{ij} \land t_{ij}$ is unsatisfiable. Similarly, the gates are complementary subject to domain constraints iff formula $\phi \land \psi \land \tau_{ij} \land \neg t_{ij}$ is unsatisfiable. Both checks can be done incrementally, using the same formula and a single auxiliary variable t_{ij}. Comparing every gate pair could be impractical for larger networks as the number of unsatisfiability checks is quadratic in the number of gates. While all the models we test in the experiments are small enough to be fully simplified, larger networks could limit the comparisons per gate only to those residing in the same layer or in two consecutive layers; our results suggest that this often achieves reductions close to those obtained by exhaustive comparison.

Finally, we can also find all the gates that degenerate into a constant false or true under domain constraints. For that, we can use the *backbone* computation algorithms (Janota et al., 2015) for the CNF formula ϕ encoding the LGN together with the domain constraints. Importantly, when removing a redundant or constant gate, we retain its contribution to voting in order to preserve the behavior of the original LGN.

3.6 Abductive Explanations for LGNs

There are multiple algorithms for computing a subset-minimal set of features $\mathcal{X}$ that satisfies the AXp condition (1) (Ignatiev et al., 2020). Each such algorithm iteratively (i) modifies a working set $\mathcal{X}$ of features and (ii) makes an oracle call that checks if the negation the AXp condition is unsatisfiable. At each iteration, the features currently included in $\mathcal{X}$ are fixed to the values dictated by the instance we are explaining. The other features are *free* to take any values in their domains. One can start from the complete set of features $\mathcal{F}$ and drop features one-by-one or, the other way round, start from $\emptyset$ and add features iteratively. Alternatively, one can consider dichotomy feature traversals (Izza et al., 2024; Wu et al., 2026). Our implementation drops features one by one.

Importantly, the exact AXp that is found is dependent on the order that features are traversed. A well-designed traversal order can produce smaller and subjectively more interpretable explanations. (An example of how feature order may impact resulting AXps is shown in Figure 1.) For explaining image-based models, we therefore traverse features sorted by each feature's value, combined with a weighted sum of its neighbours as determined by a Gaussian blur filter.



(a) (b) (c) (d) (e) (f)

Figure 1: Impact of feature traversal order on explanation why this is a 4; AXp are highlighted in **red**: (a) row traversal, left to right, top to bottom; (b) sorted by feature value; (c) sorted by feature value with weighted blur sum; (d)–(f) have the same traversals as (a)–(c), but *reversed*.

4 Experimental Results

This section briefly overviews the results of our experimental evaluation of several widely datasets. A reader is referred to Appendix A for a detailed account of the experimental setup, prototype implementation, and experimental results.[1] The experiments are performed on an Intel Xeon Gold 6150 CPU @ 2.70 GHz, with 10GB of memory allocated per instance to explain.

Implementation. Our prototype implementation is built on top of the Torchlogix framework (Gerlach et al., 2026a) used to train, simplify, and encode LGNs. Both LGN encoding and explanation

[1] Source code and complete experimental setup will be made available upon acceptance of this submission.

Table 1: Comparative model architecture and test accuracy scores for each benchmark. *LGN arch.* and *ReLU Arch.* are each a tuple of the neuron count per layer. Here, *Cmp* stands for Compas, *Adt* stands for Adult, *MND* stands for MNIST-Dense/FC, *MNC* stands for MNIST-Conv/CNN, and *FMN* stands for FashionMNIST-Conv/CNN.

Bench.	**LGN arch.**	**LGN acc.**	**ReLU arch.**	**ReLU acc.**
Cmp	(40, 2)	70.05%	(10, 2)	68.99%
Adt	(200, 2)	85.66%	(10, 2)	84.73%
MND	(1400, 10)	93.73%	(10, 10, 10)	93.76%
MNC	$(16 \times 29^2, 64 \times 28^2, 1400, 10)$	96.60%	$(4 \times 13^2, 4 \times 6^2, 20, 10)$	96.29%
FMN	$(48 \times 29^2, 192 \times 28^2, 2000, 10)$	87.73%	$(4 \times 13^2, 4 \times 6^2, 20, 10)$	86.38%

Table 2: Cumulative simplification of CNF reduction for FashionMNIST-Conv benchmark model. *Nodes* represents the number of logic gates contained in the intermediate logical circuit representation of the LGN. *Outputs* represents the number of non-constant outputs registered in the LGN's penultimate layer; the smaller, the better. Times measured in seconds. *Enc. Time* represents the time taken to encode the network, as a median of three samples. *Exp. Spd* represents the time taken to explain relative to the baseline (no simplification), measured by the first 25 instances of the dataset.

Cumulative Simplification	**Clauses**	**Vars**	**Outputs**	**Nodes**	**Enc. Time**	**Exp. Spd**
Raw (no simplification)	12954	5932	1833	82638	3.8	1.000
Add Torchlogix simplify	12812	5875	1832	3791	2.6	0.986
Add constrained simplify	11885	5874	1831	3543	7238.8	0.974
Add PySAT simplify	11885	5607	1831	3543	7091.5	0.968
Add CNF preprocessing	8664	5607	1831	3543	7186.8	0.914

of their decisions utilize the functionality of the PySAT framework (Ignatiev et al., 2018; 2024). In particular, k-modulo-totalizer (Morgado et al., 2014) encoding is used for encoding the voting mechanism and Glucose 3 (Audemard et al., 2013) is used as the underlying SAT oracle. The MaxSAT component of the implementation was adapted from XReason, originally developed for (Ignatiev et al., 2022). For the purposes of comparison, VeriX+ (Wu et al., 2026) was used to generate AXps for ReLU-based neural networks, using the exact models provided by the authors.

Benchmarks. The experiments are run across 4 standard datasets comprised of two tabular datasets: Compas (ProPublica) and Adult (Kohavi, 1996); and two image-based datasets: MNIST (LeCun et al., 2010) and FashionMNIST (Xiao et al., 2017). The tabular datasets are trained on dense models, and the image-based datasets are trained on convolutional models, with an additional dense model trained for MNIST to achieve parity with the respective models used for VeriX+. Comparative accuracy of the trained models is shown in Table 1. Given a benchmark dataset, our experiment is set to randomly select 200 instances and explain the corresponding decision made by the trained LGN and ReLU-based NN. (The same instances are used in both cases.) Note that VeriX+ originally aims at computing distance-restricted explanations that hold within an ϵ-ball of interest; for a fair comparison, we set its distance parameter $\epsilon = 1$, i.e. we seek explanations valid in the feature space.

LGN Simplification. All the simplifications applicable to LGNs and their CNF representations described in Section 3.5 have been applied in order to reduce the numbers of variables and clauses in final Boolean formulas given to the LGN explainers. Table 2 exemplifies the result of cumulatively adding them in the order they are executed in our prototype in the particular case of FashionMNIST, which is the largest benchmark among those we consider. While *all the techniques positively impact* explanation extraction time, a few techniques stand out. First, Torchlogix's own circuit simplification gets rid of more than 95% of gates. Second, the most notable reduction in the number of clauses results from constrained simplification and CNF preprocessing while the largest reduction in the number of variables stems from PySAT's internal formula simplification. The following results are obtained on the fully simplified LGNs.

Table 3: Explanation computation time by method. See Figure 2 for detailed plots.

Benchmark	SAT			MaxSAT			VeriX+		
	min.	avg.	max.	min.	avg.	max.	min.	avg.	max.
Compas	0.004	0.004	0.020	0.008	0.009	0.013	1.9	2.7	4.8
Adult	0.006	0.007	0.016	0.027	0.036	0.097	7.5	16.7	79.7
MNIST-FC	0.840	3.149	49.552	47.460	74.189	108.721	95.5	458.4	2105.5
MNIST-CNN	1.200	7.345	121.585	63.446	85.070	127.402	297.8	930.1	3022.5
FashionMNIST	2.897	5.416	33.309	184.707	230.308	296.545	135.2	621.0	2732.6

Table 4: Explanation size by method. See Figure 2 for detailed plots.

Benchmark	(total # feat.)	LGN			ReLU			Avg. ratio
		min.	avg.	max.	min.	avg.	max.	
Compas	(8)	3	4.6	6	1	3.7	7	**1.24**
Adult	(12)	4	7.1	11	1	7.3	12	**0.97**
MNIST-FC	(784)	131	172.7	244	489	604.0	695	**0.29**
MNIST-CNN	(784)	219	270.3	319	400	565.8	670	**0.48**
FashionMNIST	(784)	299	400.5	498	498	621.9	689	**0.63**

Explanation Time. In terms of explanation time, all tested benchmarks demonstrate a clear trend observable in Table 3: both SAT- or MaxSAT-based explainers of our fully simplified LGN encodings outperform their ReLU-aiming counterpart *by a few orders of magnitude*. This evidently demonstrates the advantage of LGNs (and the existing machinery for their simplification) over ReLU-based neural networks trained with equivalent test accuracy. Also observe that the SAT-based explainer is the fastest outperforming MaxSAT on all benchmarks, which can perhaps not surprising as (1) each relevance check in MaxSAT requires a sequence of SAT calls instead of one such call and (2) cardinality constraints admit cheap CNF representations.

Explanation Size. Interestingly, LGNs demonstrate advantage not only in terms of the explanation computation time, but also in terms of the resulting explanation size. Given that explanation size is often directly correlated with our understanding of the model's operation, this observation may be crucial in interpretability-critical settings, where succinct reasons behind the model's decisions are sought. Table 4 contrasts the size of explanations computed for LGN's decisions with those computed for ReLU-based neural nets. While for tabular datasets the average explanation size is comparable, one can observe a stark difference for image benchmarks, where LGN's explanations are on average half the size (or less) of those of ReLU nets. Although the phenomenon is not easy to explain, shorter reasons for model's behavior across many instances suggest better generalizability.

5 Conclusions

Logic Gate Networks provide a promising alternative to conventional neural networks, with advantages extending beyond fast inference. In this paper, we showed that they are considerably easier to explain than their ReLU counterparts of comparable size and predictive accuracy. Namely, we developed SAT- and MaxSAT-based approaches for computing formal abductive explanations of LGNs, including networks with convolutional and pooling layers. Our approach builds on compact CNF encodings of the network facilitated by the well-developed apparatus of the modern SAT technology. Interestingly, the results also suggest that LGNs can yield smaller explanations, potentially providing an additional advantage for interpretability.

Our findings highlight the importance of model architecture not only for predictive performance or inference cost, but also for the computational cost of reliable formal explanation. Furthermore, our powerful constrained LGN simplification can benefit practical LGN implementations if incorporated in their simplification functionality. Future work includes evaluating the approach in practical settings where the advantages of logic gate networks can prove most beneficial, including tackling complex neuro-symbolic tasks (Yang et al., 2020; Li et al., 2023; Manhaeve et al., 2026).

5.1 Reproducibility Statement

To ensure the reproducibility of our experiments, we have included an anonymized copy of our prototype code in the supplementary materials, including the model weights used for all benchmarks. Instructions for obtaining the main results are included in separate READMEs for our modified versions of Torchlogix and VeriX+. Our LGN results can also be reproduced using Torchlogix, PySAT, XReason, publicly available datasets, and the methods described in this paper. Our VeriX+ results are easily reproducible with the modest changes described in Section 4 and Appendix A. VeriX+ is publicly available from `https://github.com/minwu-cs/VeriX-plus`. Tables 8–15 describe the exact architectures used for all models tested. Finally, we will release fully documented code upon acceptance of the submission.

References

ACM. Fathers of the deep learning revolution receive ACM A.M. Turing award. `http://tiny.cc/9plzpz`, 2018.

Taegun An, Dohun kim, Haebeom Lee, and Changhee Joo. On the depth scalability of logic gate networks, 2026. URL `https://arxiv.org/abs/2607.21633`.

Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *SAT*, pp. 309–317, 2013.

Adrien Benamira, Thomas Peyrin, Trevor Yap, Tristan Guérand, and Bryan Hooi. Truth table net: Scalable, compact & verifiable neural networks with a dual convolutional small Boolean circuit networks form. In *IJCAI*, pp. 13–21, 2024.

Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (eds.). *Handbook of Satisfiability: Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 2021. IOS Press.

Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer Encoding: One Hot Way to Resist Adversarial Examples. In *ICLR*, 2018.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *KDD*, pp. 785–794, 2016.

Peter Clark and Robin Boswell. Rule induction with CN2: some recent improvements. In *EWSL*, pp. 151–163, 1991.

Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. In *ECAI*, pp. 712–720, 2020.

Katarzyna Fojcik, Renaldas Zioma, and Jogundas Armaitis. Lilogic net: Compact logic gate networks with learnable connectivity for efficient hardware deployment, 2026. URL `https://arxiv.org/abs/2511.12340`.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

Lino Gerlach, Thore Gerlach, Elliott Kauffman, and Liv Våge. torchlogix, 2026a. https://pypi.org/project/torchlogix/0.1.1/.

Lino Gerlach, Thore Gerlach, Liv Våge, Elliott Kauffman, and Isobel Ojalvo. Warp logic neural networks, 2026b. URL `https://arxiv.org/abs/2602.03527`.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5): 93:1–93:42, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778. IEEE Computer Society, 2016.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NIPS*, pp. 4107–4115, 2016.

Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Inf. Process. Lett.*, 5(1):15–17, 1976.

Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pp. 428–437, 2018.

Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models. In *AAAI*, pp. 1511–1519, 2019.

Alexey Ignatiev, Nina Narodytska, Nicholas Asher, and Joao Marques-Silva. From contrastive to abductive explanations and back again. In *AI*IA*, pp. 335–355, 2020.

Alexey Ignatiev, Yacine Izza, Peter J. Stuckey, and Joao Marques-Silva. Using MaxSAT for efficient explanations of tree ensembles. In *AAAI*, pp. 3776–3785, 2022.

Alexey Ignatiev, Zi Li Tan, and Christos Karamanos. Towards universally accessible SAT technology. In *SAT*, pp. 4:1–4:11, 2024.

Yacine Izza, Xuanxiang Huang, António Morgado, Jordi Planes, Alexey Ignatiev, and Joao Marques-Silva. Distance-restricted explanations: Theoretical underpinnings & efficient implementation. In *KR*, 2024.

Mikolás Janota, Inês Lynce, and Joao Marques-Silva. Algorithms for computing backbones of propositional formulae. *AI Commun.*, 28(2):161–177, 2015.

Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, pp. 202–207, 1996.

Fabian Kresse, Emily Yu, Christoph H. Lampert, and Thomas A. Henzinger. Logic gate neural networks are good for verification, 2025. URL `https://arxiv.org/abs/2505.19932`.

Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *KDD*, pp. 1675–1684. ACM, 2016.

Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]*, 2, 2010.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Ziyang Li, Jiani Huang, and Mayur Naik. Scallop: A language for neurosymbolic programming. *Proc. ACM Program. Lang.*, 7(PLDI):1463–1487, 2023.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NIPS*, pp. 4765–4774, 2017.

Robin Manhaeve, Stefano Colamonaco, Vincent Derkinderen, Rik Adriaensen, Lucas Van Praet, Luc De Raedt, and Giuseppe Marra. DeepLog: A software framework for modular neurosymbolic AI. In *IJCAI*, pp. 8462–8465, 2026.

Joao Marques-Silva and Xuanxiang Huang. Explainability is *Not* a game. *Commun. ACM*, 67(7): 66–75, 2024.

Joao Marques-Silva and Alexey Ignatiev. Delivering trustworthy AI through formal XAI. In *AAAI*, pp. 12342–12350, 2022.

Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Wout Mommen, Lars Keuninckx, Matthias Hartmann, Werner Van Leekwijck, and Piet Wambacq. Fully trainable deep differentiable logic gate networks and lookup table networks, 2026. URL https://arxiv.org/abs/2607.09399.

António Morgado, Alexey Ignatiev, and Joao Marques-Silva. MSCG: robust core-guided MaxSAT solving. *J. Satisf. Boolean Model. Comput.*, 9(1):129–134, 2014.

Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pp. 807–814, 2010.

Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *AAAI*, pp. 6615–6624, 2018.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Deep differentiable logic gate networks. In *NeurIPS*, 2022.

Felix Petersen, Hilde Kuehne, Christian Borgelt, Julian Welzel, and Stefano Ermon. Convolutional differentiable logic gate networks. In *NeurIPS*, 2024.

ProPublica. Machine bias. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing, 2016.

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *KDD*, pp. 1135–1144, 2016.

Ronald L. Rivest. Learning decision lists. *Mach. Learn.*, 2(3):229–246, 1987.

Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining Bayesian network classifiers. In *IJCAI*, pp. 5103–5111, 2018.

G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic*, Part II:115–125, 1968.

Shengpu Wang, Yuhao Mao, Yani Zhang, and Martin Vechev. Learning compact boolean networks, 2026. URL https://arxiv.org/abs/2602.05830.

Rosina O. Weber, Adam J. Johs, Prateek Goel, and Joao Marques-Silva. XAI is in trouble. *AI Mag.*, 45(3):300–316, 2024.

Min Wu, Xiaofu Li, Haoze Wu, and Clark W. Barrett. Efficiently computing compact formal explanations. In *AAAI*, pp. 35857–35866, 2026.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing neural networks into answer set programming. In Christian Bessiere (ed.), *IJCAI*, pp. 1755–1762, 2020.

A Implementation Details

A.1 Training Details

Training hardware. Training is executed on an NVIDIA RTX 4060 Laptop GPU with 105W TGP.

Implementation. Our prototype implementation uses Torchlogix as a base to design and train LGNs. We modified Torchlogix to include compatibility with selected tabular datasets including one-hot encoding to represent categorical features. Torchlogix defaults were used for the training parameters: train/test was 80/20, learning rate 0.01, batch size 128, and models were trained for a maximum of 100,000 iterations. Torchlogix's built-in model compilation was used to increase training speed.

For FashionMNIST on VeriX+, the model was trained on exactly the same architecture used by the authors for MNIST. Tabular dataset support was also added to VeriX+ for completeness, with new classification models trained for these benchmarks. These models' accuracy metrics are presented in Table 1.

For Compas, integer *age* and categorical *charge description* were dropped from the dataset. Categorical age was kept. For Adult, integer *education number* and *final weight* were dropped. Categorical education was kept.

A.2 LGN Design for Explanation

Size is typically of little concern for LGNs; training and inference feasibly scales to logic gate counts in the millions (Petersen et al., 2024). Conversely, formal explanation time scales exponentially on problem size[2] and is not currently tractable on models of this scale. When designing an LGN we wish to explain, it is therefore much more important to optimize for the highest accuracy per logic gate possible, as any increase in model size significantly increases the computation time required for explanations, even after simplification of the model. Some further accommodations were made in our setup to improve training performance for a negligible accuracy tradeoff.

Bottlenecks. In any oracle representation of an LGN, the computation time of an explanation is dominated by the voting mechanism (described in Section 3.4). In Table 5, we observe the number of cardinality clauses increases faster on vote size than the number of logic clauses. For the 4000-vote model, multiple instances required over 60 minutes for some instances on the SAT explanation benchmark. By contrast, our main results report a maximum time of 2 minutes across all models presented there.

Although one might expect this scalability bottleneck to arise from the number of variables alone, this is unlikely to be the case. For example, our FashionMNIST-Conv model holds 5607 variables without the same penalty to explanation time.

While the exact relationship between size metrics and explanation time remains to be fully investigated, we tuned the LGNs in our paper under the assumption that smaller vote sizes lead to more efficient explanations, as the growth rate of cardinality clauses on vote size would suggest.

Model dimensions and connectivity. In the pursuit of optimizing accuracy for a limited vote size, we found learnable connections to be highly valuable, consistent with existing literature (Fojcik et al., 2026). However, the implementation available to us is costly, as it requires one learnable parameter for every possible connection (Gerlach et al., 2026a). We therefore applied the findings of Mommen et al. (2026), reducing the training memory footprint to within hardware bounds by limiting the number of candidates for each learnable input to 352. This was deemed expressive enough for the purposes of our work. Also in line with the literature (Mommen et al., 2026; An et al., 2026), preliminary tests on dense models suggested that models consisting of one dense layer matched or outperformed models with more. We therefore use no more than one dense layer in all benchmarked models.

[2] Assuming $P \neq NP$

Table 5: Logic and cardinality clauses generated from encoding a 1-layer MNIST classifier LGN with a given number of gates summed for the vote, the same as described in Table 11 with the output dimension of the hidden layer changed. No simplification steps were used on the encoding. k-modulo totalizer was used for the cardinality encoding, which we show in Section A.3 to be the most efficient cardinality encoding by explanation wallclock time. Models beneath the line required over 60 minutes to explain some instances.

Votes	Variables	Logic clauses	Cardinality clauses	Cardinality/logic ratio
800	1482	2384	24553	10.3
1200	1860	3770	43487	11.5
1400	2038	4531	54083	11.9
1600	2108	4695	66431	14.1
2400	2260	5516	122069	22.1
4000	2458	7171	246971	34.4
6000	4524	17418	440012	25.3

Convolution tuning. Unlike the models presented in Petersen et al. (2024), our LGNs do not reduce the feature dimensions during convolution, whether by pooling or with increasing per-layer stride (as in Wang et al. (2026)). Preliminary tests of convolutional layers suggested that reduction was not useful for convolutional LGNs. We speculate that pooling is less expressive than learnable connections, as merging bits via disjunction always produces some information loss. In the same vein as the previous design choices, we also limited the kernel tree depth of each layer to one, minimizing the effective depth of our LGNs. Conversely, we found it beneficial to increase the number of kernels per layer. This was balanced against maintaining a reasonable training speed, however.

Overall, the design process suggests that maximizing width and minimizing depth produces the most efficient LGNs, which corroborates the findings of more thorough investigations of these properties in the literature (Mommen et al., 2026; An et al., 2026).

Binarization tuning. Binarization settings were tuned for each benchmark, with a baseline of 1 bit per feature for image-based datasets and 5 bits per feature for tabular datasets.

For COMPAS, baseline settings were kept as the LGN's accuracy matched its ReLU counterpart.

For Adult, baseline settings limited the model's accuracy to 83.82%, which was not able to be improved by increasing the network size. Increasing to 10 bits per feature improved the bound to 84.68%, and increasing to 20 bits per feature produced 85.66% accuracy.

For MNIST, increasing the number of bits per feature negligibly improved the accuracy of the models we used. The best accuracy recorded from a convolutional model using 2 bits per feature was 96.67%; a 0.07% increase on the best 1-bit accuracy recorded.

For Fashion-MNIST, the best accuracy recorded was from a model with 3 bits per feature with 87.73% accuracy. The best 1-bit accuracy was 86.17%. 7-bit and 32-bit binarizations were also tested for FashionMNIST, with highest recorded accuracies of 87.45% and 86.99% respectively.

Parametrization and forward sampling. Raw parametrization with soft forward sampling was used for all dense models. Warp parametrization was found not to benefit these shallow models; a 4000-vote layer achieved 97.63% accuracy with raw parametrization, versus 96.28% using Warp. Warp parametrization with Gumbel-hard forward sampling is used for all convolutional models; preliminary testing suggested this combination produced the best measurable accuracy for convolutional models.

A.3 Explanation Details

Cardinality and solver choice. PySAT offers nine cardinality encodings and several solver types. All solvers and cardinality constraint combinations were benchmarked on 10 AXps and 10 CXps for the FashionMNIST-Conv model. We found the clearly best-performing combinations were Glucose or MiniSat in conjunction with modulo-totalizer or k-modulo-totalizer cardinality encoding. We therefore use Glucose 3 with k-modulo-totalizer in our main experiments.

Oracle traversal. Multiple basic variations for the ordering of pairwise comparisons per solver call were implemented, such as prioritising pairs most recently found to be satisfiable, or sorting by the maximum solving time recorded for that pair in order of fastest first with a gradual decay per call on the maximum recorded for each value. These variations did not produce observable improvements in solving time, so the naive ordering, lexicographic sorting, is used for our main experiments.

MaxSAT. The MaxSAT component of the implementation was adapted from XReason, originally developed for (Ignatiev et al., 2022). It is run with `exhaust=True`, `minz=True`, and a `trim` setting of 5.

VeriX+. Explanation of tabular datasets in VeriX+ required modification to support one-hot feature explanation. We do this by adding a disjunction constraint of one-hot assignments for each category, implemented using Marabou equations.

Feature traversal configuration. For tabular datasets on both programs, row traversal is used. Image-based datasets in our prototype use blur-weighted sorting as described in Figure 1(c), while the same datasets in VeriX+ use bounds traversal with binary search as per the program default settings.

A.4 Further Remarks on MaxSAT

In the main content of the paper, we describe the use of MaxSAT as an alternative oracle implementation for the explanation of LGNs. The experimental results then suggest it is not competitive compared to SAT. However, smaller scale tests suggest there are noteworthy scenarios where MaxSAT outperforms SAT.

We tested a MNIST-Dense model using the same architecture as Table 11, but using a LogicDense layer with 4000 gates on an explanation benchmark of 100 instances. We observed that SAT had a much faster minimum explanation time (15.2 seconds to 103.6 seconds). However, the mean explanation time was over five times *slower* than the same benchmark for MaxSAT (800.8 seconds to 137.8 seconds). The maximum time for SAT was over 24 times slower than for MaxSAT (5099.3 to 211.9 seconds). Plot comparisons are presented in Figure 3.

While the results of this test are indicative only, they support the continued inclusion of MaxSAT in future efforts to improve the scalability of formal explainability.

Table 6: ReLU network used for Compas for VeriX+. One-hot features were unrolled to one input per choice.

Layer	Input dimension	Output dimension	Description
FC_1	11	10	Fully connected linear layer
ReLU	10	10	ReLU activation
FC_2	10	2	Fully connected linear output

Table 7: Architecture used for Compas LGN, trained using raw parametrization and Gumbel-soft forward sampling.

Layer	Input dimension	Output dimension	Description
Binarization	11	35	Numerical bit depth = 5
LogicDense	35	40	Learnable candidates = 17
GroupSum	40	2	τ=1, β=0

Table 8: ReLU network used for Adult for VeriX+. One-hot features unrolled to one input per choice.

Layer	Input dimension	Output dimension	Description
FC_1	97	10	Fully connected linear layer
ReLU	10	10	ReLU activation
FC_2	10	2	Fully connected linear output

Table 9: Architecture used for Adult LGN, trained using raw parametrization and Gumbel-soft forward sampling.

Layer	Input dimension	Output dimension	Description
Binarization	97	211	Numerical bit depth = 20
LogicDense	211	200	Learnable candidates = 105
GroupSum	200	2	τ=1, β=0

Table 10: Architecture for MNIST-FC ReLU network; exactly the same one used for (Wu et al., 2026).

Layer	Input dimension	Output dimension	Description
Flatten	28×28	784	Spatial flattening of input
FC_1	784	10	Fully connected linear layer
$ReLU_1$	10	10	ReLU activation
FC_2	10	10	Fully connected linear layer
$ReLU_2$	10	10	ReLU activation
FC_3	10	10	Fully connected linear output

Table 11: Architecture used for MNIST-Dense LGN, trained using raw parametrization and Gumbel-soft forward sampling.

Layer	Input dimension	Output dimension	Description
Binarization	28×28	28×28	Numerical bit depth = 1
Flatten	28×28	784	Spatial flattening of input
LogicDense	784	1400	Learnable candidates = 352
GroupSum	1400	10	τ=1, β=0

Table 12: Architecture for MNIST-CNN ReLU network; exactly the same one used for (Wu et al., 2026).

Layer	Input dimension	Output dimension	Description
Convolution 2D	28×28	$13 \times 13 \times 4$	Kernel size 3×3, stride 2
Convolution 2D	$13 \times 13 \times 4$	$6 \times 6 \times 4$	Kernel size 3×3, stride 2
Flatten	$6 \times 6 \times 4$	144	Spatial flattening of input
FC_1	784	10	Fully connected linear layer
ReLU	10	10	ReLU activation
FC_2	10	10	Fully connected linear output

Table 13: Architecture used for MNIST-Conv CLGN, trained using Warp parametrization and Gumbel-hard forward sampling.

Layer	Input dimension	Output dimension	Description
Binarization	28×28	28×28	Numerical bit depth = 1
LogicConv2d_1	28×28	$16 \times 29 \times 29$	Depth=1, stride=1, receptive field 2×2, pad 1
LogicConv2d_2	$16 \times 29 \times 29$	$64 \times 28 \times 28$	Depth=1, stride=1, receptive field 4×4, pad 1
Flatten	$64 \times 28 \times 28$	50176	Spatial flattening of logic feature maps
LogicDense	50176	1400	Learnable candidates = 352
GroupSum	1400	10	τ=1, β=0

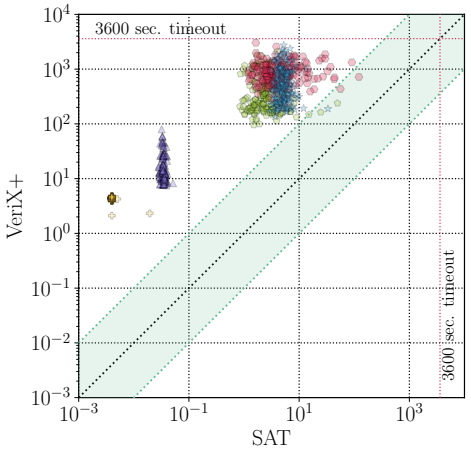
Table 14: Architecture used for FashionMNIST ReLU network, which is the same as MNIST-CNN but trained on FashionMNIST.

Layer	Input dimension	Output dimension	Description
Convolution 2D	28×28	$13 \times 13 \times 4$	Kernel size 3×3, stride 2
Convolution 2D	$13 \times 13 \times 4$	$6 \times 6 \times 4$	Kernel size 3×3, stride 2
Flatten	$6 \times 6 \times 4$	144	Spatial flattening of input
FC_1	784	10	Fully connected linear layer
ReLU	10	10	ReLU activation
FC_2	10	10	Fully connected linear output

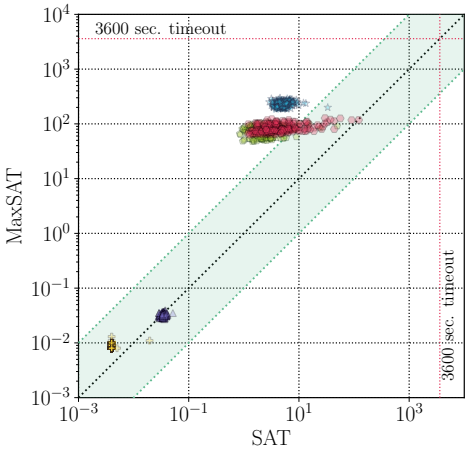
Table 15: Architecture used for FashionMNIST-Conv CLGN, trained using Warp parametrization and Gumbel-hard forward sampling.

Layer	Input dimension	Output dimension	Description
Binarization	28×28	$3 \times 28 \times 28$	Numerical bit depth = 3
LogicConv2d_1	$3 \times 28 \times 28$	$48 \times 29 \times 29$	Depth=1, stride=1, receptive field 2×2, pad 1
LogicConv2d_2	$48 \times 29 \times 29$	$192 \times 28 \times 28$	Depth=1, stride=1, receptive field 4×4, pad 1
Flatten	$64 \times 28 \times 28$	150528	Spatial flattening of logic feature maps
LogicDense	150528	2000	Learnable candidates = 352
GroupSum	2000	10	τ=1, β=0

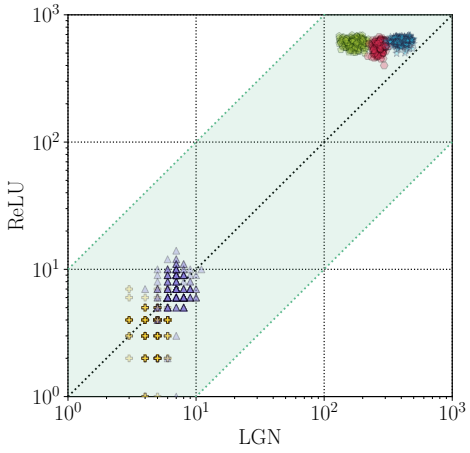
B SUPPLEMENTARY FIGURES



(a) SAT vs. VeriX+ runtime.

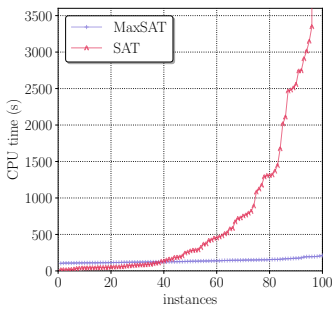


(b) SAT vs. MaxSAT runtime.

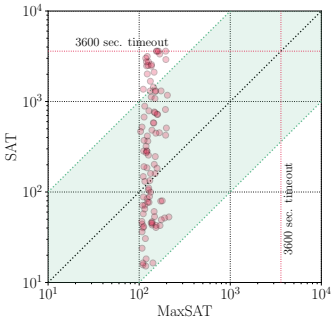


(c) Explanation size comparison.

Figure 2: Time and size results from all benchmarks. Plotted on logarithmic scale. ✚: Compas. ▲: Adult. ⬟: MNIST-Dense. ⬢: MNIST-Conv. ★: FashionMNIST.



(a) Runtime cactus plot.



(b) Runtime scatter plot.

Figure 3: Comparison of SAT and MaxSAT on an MNIST-Dense LGN with 4000 votes.