UNIVERSAL BOOLEAN REASONING IS CERTIFIABLE VIA STOCHASTIC BOOLEAN CIRCUITS

Anonymous authors
Paper under double-blind review

ABSTRACT

The proliferation of agentic systems has thrust the reasoning capabilities of AI into the forefront of contemporary machine learning. While it is known that there *exist* neural networks which can reason through any Boolean task $f : \{0,1\}^B \to \{0,1\}$, in the sense that they emulate Boolean circuits composed of binary, single-output Boolean gates, trained models have been repeatedly demonstrated to fall short of these theoretical ideals. This raises the question: *Can one exhibit a deep learning model which always* ***certifiably*** *reasons and can* ***universally*** *reason through any Boolean task?* Moreover, such a model should ideally require few parameters to solve simple Boolean tasks. We answer this question affirmatively by exhibiting a deep learning architecture which parameterizes distributions over Boolean circuits with the guarantee that, for every parameter configuration, a sample is almost surely a valid Boolean circuit (and hence admits an intrinsic circuit-level certificate). We then prove a universality theorem: for any Boolean $f : \{0,1\}^B \to \{0,1\}$, there exists a parameter configuration under which the sampled circuit computes f with arbitrarily high probability. When f is an $\mathcal{O}(\log B)$-junta, the required neuron count and circuit-description size scale near-linearly with the input dimension B. Empirically, we evaluate the induced probabilistic predictor on truth-table realization benchmarks. Under neuron-matched budgets, our stochastic Boolean circuits achieve strong predictive performance while retaining intrinsic circuit-level stochastic semantics. Matched MLPs remain competitive as unconstrained predictors, but typically do not yield Boolean-valued hidden representations. This highlights the central tradeoff of this work: unconstrained real-valued prediction versus probabilistic circuit-structured reasoning.

1 INTRODUCTION

With the recent boom in agentic models that act over extended horizons (Shinn et al., 2023; Wang et al., 2024), AI-based systems that orchestrate tools and structured search (Shen et al., 2023; Schick et al., 2023; Yao et al., 2023a;b), and iterative self-improvement techniques for refining solutions or bootstrapping rationales (Madaan et al., 2023; Wang et al., 2023; Zelikman et al., 2022; Chen et al., 2025; Song et al., 2025b), (purported) AI reasoning has become commonplace in modern machine-learning pipelines, increasingly influencing end-user decisions. This burst of AI reasoning models rests on well-forged connections between deep learning and complexity theory from: the Turing machine lens (Siegelmann, 2012; Pérez et al., 2019; Chung & Siegelmann, 2021; Mali et al., 2021; Bournez et al., 2025) for recursive/looped networks; the Boolean (Maass et al., 1991; Siu et al., 1993; Merrill et al., 2022; Merrill & Sabharwal, 2023; Chiang, 2024) and general circuit complexity lenses (Lowd & Domingos, 2008; Kratsios et al., 2025) for non-recursive architectures (e.g., MLPs and transformers); as well as connections to mathematical logic (Blumer et al., 1989; Karpinski & Macintyre, 1997; Margulies & Morton, 2016; Cook & Nguyen, 2010). Broadly speaking, these results show that there **exist** deep learning models which can, in principle, be configured to reason through arbitrary tasks; when reasoning is understood as exactly emulating an algorithm, or equivalently a *circuit*, computing some (Boolean) function.

However, existential realizability alone does not guarantee Boolean gate structure throughout the parameter space. Even the one-bit gate NOT already illustrates the issue. Consider a shallow

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.

ReLU-MLP $f_\theta(x) \stackrel{\text{def}}{=} \theta_3 \operatorname{ReLU}(\theta_1 x + \theta_2)$, with $\theta \in \mathbb{R}^3$. Then this model computes the NOT gate* on $\{0, 1\}$ if and only if θ belongs to the following **two**-dimensional, **non-convex** subset $\Theta_{\text{NOT}} \subseteq \mathbb{R}^3$ of the neural network's parameter space:

$$\Theta_{\text{NOT}} \stackrel{\text{def}}{=} \big\{(\theta_1, \theta_2, \theta_3) \in \mathbb{R}^3 : \theta_2 > 0,\, \theta_3 = 1/\theta_2,\, \theta_1 + \theta_2 \leq 0\big\}.$$

See Appendix A.3. This set has measure zero under any probability distribution equivalent to a Gaussian measure; thus, without additional structural bias, unconstrained continuous optimization is unlikely to land exactly on such a configuration. This concern is naturally amplified for more complicated Boolean circuits reasoning through more sophisticated Boolean tasks, since these constraints compound across gates. Thus, although there exist networks which can formally perform such computations, the learnability of the reasoning steps required to interpretably resolve Boolean reasoning problems becomes highly unlikely; cf. Anthony & Bartlett (2009); Chiang (2024); Kratsios et al. (2025).

The Existence vs. Realizability Concern. This concern is not merely theoretical: there is mounting evidence that most real-world *trained* AIs *fail* to reason through complex multi-step reasoning tasks (Keysers et al., 2020; Wei et al., 2022; Zhao et al., 2024), arithmetic (Cobbe et al., 2021) or more sophisticated mathematical reasoning (Hendrycks et al., 2021), elementary deductive logic (Berglund et al., 2024), inductive reasoning (Olsson et al., 2022; Zhao et al., 2024), and several other forms of illusory reasoning peacocking (Song et al., 2025a). Experimental evidence increasingly indicates that trained deep learning pipelines produce effective memorizers of exorbitantly many training instances (Bender et al., 2021; Wu et al., 2024; Shojaee et al., 2025).

Even seemingly innocuous training-data contamination (Zou et al., 2025; Chaudhari et al., 2026) or adversarial prompt attacks (Liu et al., 2024; Rajeev et al., 2025) are known to devastate these models' reasoning. This heaps on the evidence that AIs *trained* on *data* often *fail* to reason and, even when they appear to reason, their chain-of-thought can be broken by adversarial prompt injection. Thus, there is a very real **AI safety concern**; namely, how can real-world AIs (not those configured within a mathematical vacuum e.g. Kratsios et al. (2025)) be guaranteed to certifiably reason. This leads to a natural question:

> How can one construct a neural network architecture which always admits a *certifiable*, *universal*, and *efficient* circuit-level interpretation, while remaining competitive as a probabilistic predictor on task-wise propositional reasoning benchmarks?

More precisely, we seek three guarantees:

(i) **Validity for every parameter setting:** for every parameter configuration, the induced random function is almost surely a valid circuit of binary, single-output Boolean gates.

(ii) **Existential universality:** for every Boolean function f and every failure probability $\delta > 0$, there exist an SBC architecture and parameter configuration whose sampled circuit computes f on the entire Boolean cube with probability at least $1 - \delta$.

(iii) **Efficient realization of logarithmic juntas:** if f is an S-junta with $S \leq C \log B$, the construction has polynomial neuron and circuit-description size $\mathcal{O}(B^C \log B)$; when $0 < C \leq 1$, this becomes near-linear $\mathcal{O}(B \log B)$.

Our experiments separately investigate whether a differentiable relaxation of a restricted SBC implementation can be optimized effectively on task-wise truth-table realization problems.

1.1 Contribution

We introduce trainable stochastic Boolean circuits (SBCs), distributions over circuits composed of binary, single-output Boolean gates. For every parameter setting, an SBC sample is almost surely a valid Boolean circuit (Theorem 1); suitable parameters compute any desired Boolean function with arbitrarily high probability (Theorem 2), with quantitative neuron and circuit-description size bounds for logarithmic juntas, including near-linear scaling when $S \leq \log B$ (Theorem 3). Unlike differentiable logic gate networks (Petersen et al., 2022), which learn Boolean operators on fixed

*I.e., $f_\theta(0) = 1$ and $f_\theta(1) = 0$.

random connectivity, SBC learns routing and bit-lifting, inducing a distribution over circuit topologies and gates. Experimentally, its probabilistic predictor is competitive with matched MLPs, only about 20% of whose hidden units are Boolean-valued on the Boolean cube; sampled SBC executions are Boolean-valued by construction.

Related Work on Neural Algorithmic Reasoning. We adopt the neural algorithmic reasoning (NAR) perspective (Veličković & Blundell, 2021), viewing reasoning as executing algorithms represented by Boolean circuits (Jukna, 2012). The closest architectures are differentiable logic gate networks (Petersen et al., 2022; Bührer et al., 2025), which learn Boolean gates over fixed connectivity; SBC additionally learns routing and bit-lifting, inducing a distribution over circuit topologies and gates.

2 PRELIMINARIES

2.1 SETTING AND NOTATION

We henceforth fix a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ on which all random variables are defined. All random variables will be denoted in boldface. We write $\mathbb{N}$ for the set of non-negative integers, $\mathbb{N}_+$ for the set of positive integers and, for every $N \in \mathbb{N}$ we write $\mathbb{N}_N \stackrel{\text{def}}{=} \{n \in \mathbb{N} : n \geq N\}$, $[N] \stackrel{\text{def}}{=} \{n \in \mathbb{N} : n \leq N\}$, and $[N]_+ \stackrel{\text{def}}{=} [N] \cap [1, \infty)$. For any $n, m \in \mathbb{N}_+$, each $i \in [n]_+$ and $j \in [m]_+$ and every $n \times m$ matrix A, we write $A_{i:}$ for the ith row of A and $A_{:j}$ for its jth column. All vectors are column vectors. For any matrix (resp. vector) $\|A\|_0$ counts the number of non-zero entries in A. The set of *all* binary, single-output Boolean gates, cf. Section A.2, is denoted by $\mathbb{G}^\star$. Fixing $N \in \mathbb{N}_+$, $\mathbf{1}_N \in \mathbb{R}^N$ denotes the vector of 1s. We denote the N-simplex by $\Delta_N \stackrel{\text{def}}{=} \{w \in [0,1]^N : \mathbf{1}_N^\top w = 1\}$, its relative interior is $\dot{\Delta}_N \stackrel{\text{def}}{=} \{w \in (0,1)^N : \mathbf{1}_N^\top w = 1\}$. We define the *softmax* function $\mathrm{SM}_N : \mathbb{R}^N \to \Delta_N$ as mapping any $x \in \mathbb{R}^N$ to $\mathrm{SM}_N(x) \stackrel{\text{def}}{=} \left(e^{x_n} / \sum_{m=1}^N e^{x_m}\right)_{n=1}^N$. When the dimension is clear from context, we simply write SM. For any point a, δ_a denotes the Dirac point mass at a. Throughout the paper, the standalone symbol δ denotes a failure probability.

2.2 BOOLEAN CIRCUITS OF BINARY, SINGLE-OUTPUT GATES AND THE AC^0 CLASS

Directed Acyclic Graphs. A directed graph $G = (V, E)$ is a pair of a (non-empty) set V of *nodes* and a collection of ordered pairs $E \subseteq V^2 \setminus \{(v, v)\}_{v \in V}$ representing directed edges. Given two nodes $v, w \in V$, a directed path between v and w is a sequence $(v, v_1), (v_1, v_2), \ldots, (v_k, w) \in E$ "connecting v to w," and $k + 1$ is the *length* of this directed path. Such a G is called *connected* if its underlying undirected graph is connected, i.e., any two distinct vertices can be joined by a path after ignoring edge directions. G is called *acyclic* if there are no directed paths from a node to itself. If G is connected and acyclic, we call it a *connected DAG*. Any two nodes $u, v \in V$ are partially ordered, written $u \lesssim v$, cf. (Bang-Jensen & Gutin, 2008, Proposition 2.1.3), if there is a directed path originating at u and terminating at v. The cardinality of the largest set of unordered vertices in G is the *width* of G. The *depth* of G is the length of the longest directed path in G.[†]

In contrast, an undirected graph $G_u = (V, E_u)$ is a pair of a (non-empty) set V of *nodes* and a collection of *unordered* pairs $E_u \subseteq V^2 \setminus \{(v, v)\}_{v \in V}$ representing edges. An undirected graph is called a *tree* if for every distinct $v_1, v_2 \in V$ there exists a unique path from v_1 to v_2. A *DAG* is called a directed *tree* if its underlying undirected graph is a tree. The set of nodes with no directed path terminating thereat are called *roots/input nodes* and the set of nodes for which there is no directed path originating thereat are called *leaves/output nodes*. A binary tree is a tree in which every non-input node $v \in V$ has at most two distinct nodes $u, w \in V$ such that $(u, v), (w, v) \in E$.

Boolean Circuits. A *binary, single-output Boolean gate* is a function $g : \{0,1\}^2 \to \{0,1\}$. Here, *single-output* means that the gate computes one Boolean value; this value may be reused by arbitrarily many downstream gates, so its graph-theoretic fan-out is unrestricted. Examples include AND, OR, and the lifted NOT gate $\mathrm{NOT}(x_1, x_2) \stackrel{\text{def}}{=} 1 - x_1$. There are exactly 16 such gates; cf. Table 3 in Appendix A.1.

[†]For the directed graph underlying the computational graph of any MLP, these notions of width and depth coincide with the usual uses of the words in MLP.

Fix $B, \Delta, \Upsilon \in \mathbb{N}_+$. In this paper, a *Boolean circuit* with B input bits, depth Δ, and width Υ refers to a triple $\mathcal{C} = (V, E, g_\cdot)$ of a DAG $G = (V, E)$ with B root/input nodes $v_{\text{in}:1}, \ldots, v_{\text{in}:B} \in V$ and one terminal/output node $v_{\text{out}} \in V$, such that: the in-degree of every non-root node is exactly 2; except for an initial *bit-lifting channel* which selects bits and their negations. Such a bit-lifting channel is formalized by a map $x \mapsto \Pi(x, \mathbf{1}_B - x)$ where Π is a matrix with 1-hot vectors as rows. The $B^\uparrow$ outputs of the bit-lifting channel are denoted by $V_{\text{lift}} = \{v_{\text{lift}:1}, \ldots, v_{\text{lift}:B^\uparrow}\}$. The set of all non-input, non-lifted nodes are called *computational nodes* and are denoted by $V_{\text{comp}} \stackrel{\text{def}}{=} V \setminus \big(\{v_{\text{in}:b}\}_{b=1}^B \cup V_{\text{lift}}\big)$. Additionally, $g_\cdot = (g_v)_{v \in V_{\text{comp}}}$ is a family of binary, single-output Boolean gates. The set of all such Boolean circuits[‡] is denoted by $\text{CIRCUIT}^B_{\Delta,\Upsilon}$. Since $\mathcal{C}$ has depth Δ, the vertex set admits a layering $V = \bigsqcup_{l=0}^{\Delta} V_l$, with $V_0 = \{v_{\text{in}:b}\}_{b=1}^B$, $V_1 = V_{\text{lift}}$, and $V_\Delta = \{v_{\text{out}}\}$, such that every directed edge $(u, v) \in E$ satisfies $u \in V_l$ and $v \in V_{l+1}$ for some $l \in \{0, 1, \ldots, \Delta - 1\}$. Furthermore, since $\mathcal{C}$ has fan-in 2, every non-input vertex $v \in V_l$ with $l \geq 2$ has exactly two predecessors $v_{-1}, v_{-2} \in V_{l-1}$ satisfying $(v_{-1}, v), (v_{-2}, v) \in E$.

Boolean Reasoning as Computation of Boolean Functions. Applying the current AI reasoning theory lens Merrill et al. (2022); Chiang (2024), we understand reasoning through a "Boolean task" to mean the computation of a Boolean function by a Boolean circuit in $\cup_{\Delta,B,\Upsilon \in \mathbb{N}_+} \text{CIRCUIT}^B_{\Delta,\Upsilon}$; the latter being defined in the standard sense of circuit complexity theory, cf. Jukna (2012), and resting on the classical correspondence between Boolean circuits and first-order propositional logic, cf. Cook & Nguyen (2010). Fix a choice of $B^\uparrow \in \mathbb{N}_+$ (with repetition allowed) input variables from $\{x_b\}_{b=1}^B$ and their negations $\{(1 + x_b) \mod 2\}_{b=1}^B$, with $B^\uparrow \leq \Upsilon$, equivalently encoded by a $B^\uparrow \times 2B$ row-stochastic matrix Π. Then every Boolean circuit $\mathcal{C} \in \text{CIRCUIT}^B_{\Delta,\Upsilon}$ induces a Boolean function $f_\mathcal{C} : \{0,1\}^B \to \{0,1\}$ defined recursively as:

$$\begin{aligned} f_\mathcal{C}(x) &\stackrel{\text{def}}{=} f_\Delta(x) \\ f_l(x) &\stackrel{\text{def}}{=} \big(g_v\big(f_{l-1}(x)_{v_{-1}}, f_{l-1}(x)_{v_{-2}}\big)\big)_{v \in V_l}, \qquad l = 2, \ldots, \Delta, \\ f_1(x) &\stackrel{\text{def}}{=} x_1 \\ x_1 &\stackrel{\text{def}}{=} \Pi \begin{pmatrix} x \\ \mathbf{1}_B - x \end{pmatrix}, \end{aligned} \tag{1}$$

where v_{-1} and v_{-2} refer to the indices on the vector $f_{l-1}(x)$ corresponding to the two parent nodes of v. We say that a Boolean function $f : \{0,1\}^B \to \{0,1\}$ is *computable* if there exists a Boolean circuit $\mathcal{C}$ such that its induced function $f_\mathcal{C}$ satisfies

$$f_\mathcal{C}(x) = f(x) \qquad \text{for every } x \in \{0,1\}^B.$$

There are several *complexity classes* in the theory of non-uniform models of computing/circuit complexity, e.g. AC^k, NC^k, TC^k, etc., for $k \in \mathbb{N}$. These do not describe which Boolean functions are, or are not, computable, but rather classify how hard it is to compute a given computable Boolean function by bounding the depth of the circuits computing it, depending on how "powerful" the allowed gates are. For instance, AC^k allows unbounded fan-in AND, OR, and NOT gates; NC^k restricts to bounded fan-in AND / OR / NOT gates, corresponding to highly parallelizable circuit families; and TC^k allows the use of more powerful gates such as threshold (majority) gates, which are not computable by more elementary Boolean gate sets; cf. e.g. Håstad (1986) in polynomial time. A key new set of properties that our model manages to ensure simultaneously is that it almost surely samples a random element of $\text{CIRCUIT}^B_{\Delta,\Upsilon}$ for prescribed Δ, Υ, and B (Theorem 1), and that every such circuit is sampleable by our universal model (Theorem 2). Moreover, we may train its weights and biases by differentiable optimization so as to guide it toward any given target circuit.

2.3 A Test for Latent Boolean Reasoning

We next pose a diagnostic question: *"what does it mean for a non-Boolean function to latently perform Boolean reasoning?"* If we can formalize this, then we can easily test whether neurons in standard neural networks, e.g. MLPs, are performing a latent form of Boolean reasoning. Whatever

[‡]Any binary, single-output Boolean gate is allowable in this class; one may also consider the subclass where only AND, OR, and NOT gates are used, with no essential change to the theory.

our definition may be, it should be aligned with the ideas behind reasoning probes, e.g. (Alain & Bengio, 2016; Craven & Shavlik, 1995; Dai et al., 2022), and it should always be satisfied for our idealized reasoners; namely, Boolean circuits.

We say that a function f from the Boolean cube $\{0,1\}^B$ to some Euclidean space admits a "Boolean neuronal representation," or interpretation, if each component of f can be understood as assigning Boolean values to the entries of x; thus, each component can be interpreted as a neuron. For this to be possible, of course, each potential neuron of f, i.e. each component thereof, cannot assign more than two distinct values to the members of the Boolean cube; otherwise, there have to be more than two truth values which is impossible.

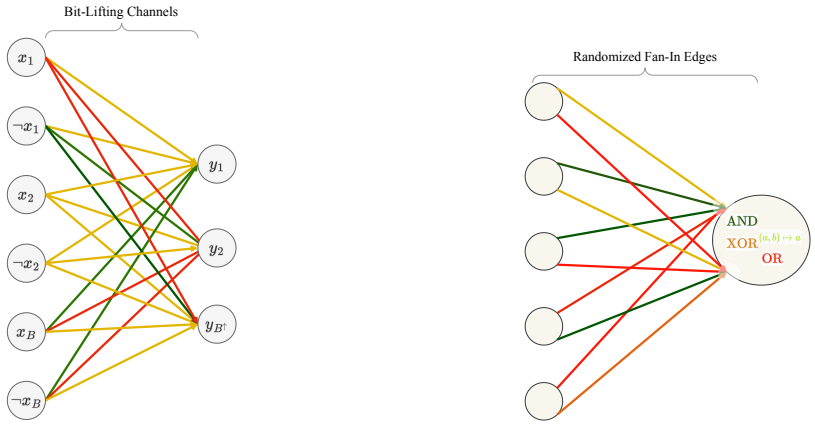
Definition 1 (Boolean Neuronal Representability)**.** Let $B_{\text{in}}, B_{\text{out}} \in \mathbb{N}_+$. A function $f : \{0,1\}^{B_{\text{in}}} \to \mathbb{R}^{B_{\text{out}}}$ is said to admit a Boolean neuronal representation if, for every $i \in [B_{\text{out}}]_+$ there is a map $\mathcal{R}_i : \mathcal{X}_i \to \{0,1\}$ which is injective; where $\mathcal{X}_i \stackrel{\text{def}}{=} p_i \circ f(\{0,1\}^{B_{\text{in}}})$ and p_i denotes the ith coordinate projection. Whenever such $\mathcal{R}_1, \ldots, \mathcal{R}_{B_{\text{out}}}$ exist, f is said to be *Boolean neuronally representable (BNR)*. Any map $\mathcal{R} : \mathbb{R}^{B_{\text{out}}} \to \{0,1\}^{B_{\text{out}}}$ is called a Boolean neuronal representation of f, if

$$\mathcal{R}(f(x)) = \big(\mathcal{R}_i(f(x)_i)\big)_{i=1}^{B_{\text{out}}} \qquad \text{for all } x \in \{0,1\}^{B_{\text{in}}}.$$

By construction, every Boolean function is BNR. More generally, $\{0,1\}$-valued functions such as MLPs with the neuroscience-inspired *threshold activation function*, e.g. McCulloch & Pitts (1943), also achieve a BNR density of 1 (see Definition 3). In contrast, even simple modern neurons, e.g. ReLU neurons, easily fail to be BNR (see Example 3 and Proposition 1).

3 The Stochastic Boolean Circuit Model

Our Stochastic Boolean Circuit model (Figure 1) is assembled from three components: a parameterized distribution over binary, single-output Boolean gates; a distribution over pairs of edges; and a stochastic bit-lifting channel.



(a) ***Stochastic Bit-Lifting Channels*****:** The first layer of our model employs stochastic bit-lifting channels (cf. (3)), which select $B^\uparrow$ among the B input bits and their negations (duplicates allowed).

(b) ***Randomized Boolean Neuron*****:** Exactly two edges (fan-in 2) are selected from the nodes of the previous layer (cf. (4)); then, a Boolean gate is randomly selected (cf. (5)) and the output is computed.

Figure 1: Building blocks of the SBC: stochastic bit-lifting selects literals, while each internal node samples two parents and a fan-in-2 Boolean gate. Red edges/gates indicate lower selection probability, whereas green edges/gates indicate higher selection probability.

We now rigorously define the components of the Stochastic Boolean Circuit model.

3.1 Model Components: Stochastic Bit-Lifting Channels

Let $B, B^{\uparrow} \in \mathbb{N}_+$, and let $W \in \mathbb{R}^{B^{\uparrow} \times 2B}$. We define the stochastic bit-lifting channel $\mathcal{L}(\cdot|W) : \mathbb{R}^B \to \mathbb{R}^{B^{\uparrow}}$ for every $x \in \mathbb{R}^B$ by

$$\mathcal{L}(x|W) \stackrel{\text{def}}{=} \mathrm{SM}(W) \left(\begin{pmatrix} I_B \\ -I_B \end{pmatrix} x + \begin{pmatrix} \mathbf{0}_B \\ \mathbf{1}_B \end{pmatrix} \right) \tag{2}$$

where the softmax function SM is applied row-wise to the matrix W. We use $\mathcal{L}(\cdot|W)$ to parametrize the distribution of the following random variable. Specifically, let $\mathbf{X}^{\uparrow:W}$ be a $C(\{0,1\}^B, \{0,1\}^{\uparrow B})$-valued random variable whose i^{th} coordinate $\mathbf{X}_i^{\uparrow:W}$ has distribution, given for any $x \in \{0,1\}^B$, by

$$\mathbf{X}_i^{\uparrow:W}(x) \sim \sum_{j=1}^{B} \left(\mathrm{SM}_{2B}(W_{i:}) \right)_j \delta_{x_j} + \sum_{j=1}^{B} \left(\mathrm{SM}_{2B}(W_{i:}) \right)_{j+B} \delta_{\neg x_j}. \tag{3}$$

3.2 Model Components: Randomized Boolean Neuron

3.2.1 Edges: Trainable Random Fan-in Edges

For every $B \in \mathbb{N}_+$ and every temperature parameter $\eta > 0$, define the operation $\mathrm{S}_\eta : \mathbb{R}^B \times \mathbb{R}^B \to \mathbb{R}^B \times \mathbb{R}^B$ as mapping any $(W_1, W_2) \in \mathbb{R}^B \times \mathbb{R}^B$ to

$$\mathrm{S}_\eta \left(W_1, W_2 \right) \stackrel{\text{def}}{=} \Big(\mathrm{SM}(\eta W_1), \mathrm{SM} \left(\eta(1 - \mathrm{SM}(\eta W_1)) \odot \mathrm{SM}(\eta W_2) \right) \Big) \tag{4}$$

where $\odot$ denotes the componentwise (Hadamard) product on pairs of vectors in $\mathbb{R}^B$. For $i \in \{1,2\}$, $S_\eta(W_1, W_2)_i$ represents the distribution in $\dot{\Delta}_B$ for input i of a given Boolean fan-in 2 gate.

3.2.2 Gates: Trainable Stochastic Boolean Gates

Let $g_1, \ldots, g_{16}$ enumerate all Boolean gates $\{0,1\}^2 \to \{0,1\}$, and fix a smooth corner-exact interpolation $\sigma = (\sigma_1, \ldots, \sigma_{16}) : \mathbb{R}^2 \to \mathbb{R}^{16}$ satisfying $\sigma_i(x) = g_i(x)$ for all $x \in \{0,1\}^2$ and $i \in [16]_+$ (see Appendix D for an explicit construction). Given gate logits $W_\Sigma \in \mathbb{R}^{16}$, draw $\mathbf{X}^{W_\Sigma} \in \{e_1, \ldots, e_{16}\}$ with $\mathbb{P}(\mathbf{X}^{W_\Sigma} = e_i) = \left(\mathrm{SM}_{16}(W_\Sigma) \right)_i$, and define

$$\boldsymbol{\sigma}^{W_\Sigma}(x) \stackrel{\text{def}}{=} \sum_{i=1}^{16} \mathbf{X}_i^{W_\Sigma} \sigma_i(x). \tag{5}$$

Since σ_i agrees with g_i on the Boolean corners, $\boldsymbol{\sigma}^{W_\Sigma}|_{\{0,1\}^2}$ is always one of the 16 Boolean gates.

3.2.3 Random Circuit Network Recursion

Fix a depth parameter $\Delta \in \mathbb{N}_+$, widths $(B_{\text{in}}, \ldots, B_{\text{out}}) \in \mathbb{N}_+^{\Delta+1}$, and a temperature parameter $\eta > 0$. For every layer $l \in [\Delta - 1]_+$ and every set of node weights $W^{(l:\sigma)} \in \mathbb{R}^{B_{l+1} \times 16}$, we define the **stochastic Boolean gate**/activation function $\boldsymbol{\sigma}^{W^{(l:\sigma)}} : \mathbb{R}^{B_{l+1}} \times \mathbb{R}^{B_{l+1}} \to \mathbb{R}^{B_{l+1}}$ as mapping every $(x, y) \in \mathbb{R}^{B_{l+1}} \times \mathbb{R}^{B_{l+1}}$ to

$$\boldsymbol{\sigma}^{W^{(l:\sigma)}}(x, y) \stackrel{\text{def}}{=} \left(\boldsymbol{\sigma}^{W_j^{(l:\sigma)}}(x_j, y_j) \right)_{j=1}^{B_{l+1}}. \tag{6}$$

Then, $\boldsymbol{\sigma}^{W^{(l:\sigma)}}$ specifies each gate at every node of the stochastic Boolean circuit implementing our model as parameterized by the trainable matrix $W^{(l:\sigma)}$. At every layer $l \in [\Delta - 1]_+$ the edges in our stochastic Boolean circuit are defined by the $B_{l+1} \times B_l$ matrices

$$\boldsymbol{E}^{(l:1)} \stackrel{\text{def}}{=} \begin{bmatrix} \mathrm{SM}(\eta W_1^{(l:1)}) \\ \vdots \\ \mathrm{SM}(\eta W_{B_{l+1}}^{(l:1)}) \end{bmatrix} \text{ and } \boldsymbol{E}^{(l:2)} \stackrel{\text{def}}{=} \begin{bmatrix} \mathrm{SM} \left(\eta \big(1 - \mathrm{SM}(\eta W_1^{(l:1)})\big) \odot \mathrm{SM}(\eta\, W_1^{(l:2)}) \right) \\ \vdots \\ \mathrm{SM} \left(\eta \big(1 - \mathrm{SM}(\eta W_{B_{l+1}}^{(l:1)})\big) \odot \mathrm{SM}(\eta\, W_{B_{l+1}}^{(l:2)}) \right) \end{bmatrix}, \tag{7}$$

where $\odot$ denotes the componentwise (Hadamard) product on pairs of vectors in $\mathbb{R}^{B_l}$. The factor $1 - \mathrm{SM}(\eta W_1)$ in $\boldsymbol{E}^{(l:2)}$ downweights the parent selected with high probability by the first edge distribution, encouraging the two parents to be distinct. Let $\boldsymbol{E}^{(l:1:e)}$ and $\boldsymbol{E}^{(l:2:e)}$ denote the corresponding sampled adjacency matrices whose row laws are specified by (7) (i.e., for each $j \in [B_{l+1}]_+$, the j-th row of $\mathbf{E}^{(l:1:e)}$ and $\mathbf{E}^{(l:2:e)}$ is drawn as a one-hot vector in $\{e_1, \ldots, e_{B_l}\}$ with mean given by the jth row of $E^{(l:1)}$ and $E^{(l:2)}$, respectively). We may now formally define our SBC model.

Definition 2 (The Stochastic Boolean Circuit Model)**.** Fix $B_{\mathrm{in}}, B^{\uparrow}, \Delta, B_{\mathrm{out}} \in \mathbb{N}_+$. The Stochastic Boolean Circuit is a map $\hat{\boldsymbol{f}} : \Omega \times \{0,1\}^{B_{\mathrm{in}}} \to \{0,1\}^{B_{\mathrm{out}}}$ defined by iteratively mapping any $x \in \{0,1\}^{B_{\mathrm{in}}}$ to[§]

$$\begin{aligned} \hat{\boldsymbol{f}}(x) &\stackrel{\mathrm{def}}{=} \boldsymbol{x}^{(\Delta)} \\ \boldsymbol{x}^{(l+1)} &\stackrel{\mathrm{def}}{=} \boldsymbol{\sigma}^{W^{(l:\sigma)}}\big(\boldsymbol{E}^{(l:1:e)}\,\boldsymbol{x}^{(l)}, \boldsymbol{E}^{(l:2:e)}\,\boldsymbol{x}^{(l)}\big) \qquad \text{for } l = 1, \ldots, \Delta - 1 \\ \boldsymbol{x}^{(1)} &\stackrel{\mathrm{def}}{=} \mathbf{X}^{\uparrow:W}(x), \end{aligned} \tag{8}$$

where the random matrices $\boldsymbol{E}^{(l:1:e)}$ and $\boldsymbol{E}^{(l:2:e)}$ are as in (7), the stochastic Boolean gates $\boldsymbol{\sigma}^{W^{(l:\sigma)}}$ are as in (6), and the stochastic bit-lifting channel $\mathbf{X}^{\uparrow:W}$ is as in (3). For $l = 1, \ldots, \Delta$, the dimensions of these objects are such that the composition defining $\hat{\boldsymbol{f}}$ is well-defined for every $(\omega, x) \in \Omega \times \{0,1\}^{B_{\mathrm{in}}}$.

Component properties. The proof of our main results relies on three component-level properties of the SBC model. First, the bit-lifting channel outputs Boolean-valued lifted literals and can be configured to repeat or omit any desired input variables or their negations; see Proposition 4. Second, the edge selector samples valid one-hot parent choices and can concentrate on any chosen pair of parents; see Proposition 5. Third, the gate selector always samples a valid binary, single-output Boolean gate and can concentrate on any specified gate; see Propositions 6 and 7.

4 Main Results

4.1 Certifiable Reasoning

We are now prepared to present our first main result which *certifies* that any possible configuration of the parameters of Stochastic Boolean Circuit must yield a distribution over circuits composed of binary, single-output Boolean gates. Consequently, every sample from such a configuration is almost surely a valid Boolean circuit composed of binary, single-output gates.

Theorem 1 (Certifiable Boolean Reasoning)**.** *Fix a depth $\Delta \in \mathbb{N}_+$, a width $\Upsilon \in \mathbb{N}_+$, and layer widths $B_0 = B_{\mathrm{in}}, \ldots, B_\Delta = B_{\mathrm{out}} \in \mathbb{N}_+$ such that $\max_{l \in [\Delta]} B_l \le \Upsilon$. Then, for any matrices W, $\{W^{(l:\sigma)}\}_{l=1}^{\Delta-1}$ and $\{\mathbf{E}^{(l:1:e)}\}_{l=1}^{\Delta-1}$, $\{\mathbf{E}^{(l:2:e)}\}_{l=1}^{\Delta-1}$ as in (8) the (random) function $\hat{\boldsymbol{f}}$ is such that*

$$\mathbb{P}\big(\hat{\boldsymbol{f}} \in \mathrm{CIRCUIT}_{\Delta,\Upsilon}^{B_{\mathrm{in}}}\big) = 1.$$

Random-circuit law and predictive marginal. Let θ denote the collection of all trainable parameters. By Theorem 1, each fixed θ induces an almost surely valid random Boolean circuit $\mathcal{C}_\theta$ with law $\mu_\theta \stackrel{\mathrm{def}}{=} \mathrm{Law}(\mathcal{C}_\theta)$; hence $\hat{\boldsymbol{f}}_\theta(x) = f_{\mathcal{C}_\theta}(x)$. The corresponding probabilistic predictor is $p_\theta(x) \stackrel{\mathrm{def}}{=} \mathbb{P}_{\mathcal{C}\sim\mu_\theta}(f_{\mathcal{C}}(x) = 1) = \mathbb{E}_{\mathcal{C}\sim\mu_\theta}[f_{\mathcal{C}}(x)]$. Thus, at the proof level, SBC is a probabilistic program whose execution traces are Boolean circuits, while prediction is given by the marginal p_θ. This exact marginal belongs to the proof-level random-circuit model. The implementation instead computes a differentiable relaxed score $\hat{p}_\theta$; Appendix F.2 details this distinction.

4.2 Universal Reasoning

Having guaranteed that every parameter setting induces a distribution supported on Boolean circuits, we now show that these distributions are universal: with sufficiently many neurons, their parameters can be configured so that a sampled circuit computes any prescribed Boolean function with arbitrarily high probability.

[§]Following standard probability theory convention, we suppress the dependence of the stochastic Boolean gates $\{\boldsymbol{\sigma}^{W^{(l:\sigma)}}\}_{l=1}^{\Delta-1}$, random output of the bit lifting channel $\mathbf{X}^{\uparrow:W}$, and random adjacency matrices $\big\{\boldsymbol{E}^{(l:j:e)}\big\}_{l=1,\,j=1}^{\Delta-1,\,2}$ on the outcome $\omega \in \Omega$.

Table 1: Truth-table exact match (EM; mean $\pm$ std across 5 seeds). "All" is the full 2000-task benchmark; "Large" restricts to $B \geq 18$. MLPs are matched against the same canonical SBC predictor.

Subset	MLP match	Avg. MLP params	MLP EM	SBC EM
Large ($B \geq 18$)	neuron	514	0.869 ± 0.007	0.930 ± 0.010
	param	983	0.927 ± 0.015	
All	neuron	405	0.957 ± 0.002	0.961 ± 0.003
	param	758	0.977 ± 0.007	

Theorem 2 (Universal Boolean Reasoning)**.** *Let $B_{\text{in}} \in \mathbb{N}_+$ and let $f : \{0,1\}^{B_{\text{in}}} \to \{0,1\}$. For every failure probability $0 < \delta \leq 1$, there exists an $\hat{\boldsymbol{f}} : \Omega \times \{0,1\}^{B_{\text{in}}} \to \{0,1\}$ with representation* (8) *such that*

$$\mathbb{P}\Big(\hat{\boldsymbol{f}}(x) = f(x)\ \forall x \in \{0,1\}^{B_{\text{in}}}\Big) \geq 1 - \delta. \tag{9}$$

Corollary 1 (Predictive marginal of a universal random circuit)**.** *In the setting of Theorem 2, let $f : \{0,1\}^{B_{\text{in}}} \to \{0,1\}$ and $E_f \stackrel{\text{def}}{=} \{\mathcal{C} : f_{\mathcal{C}}(x) = f(x)\ \forall x \in \{0,1\}^{B_{\text{in}}}\}$. If $\mu_\theta(E_f) \geq 1 - \delta$, then*

$$\|p_\theta - f\|_{\ell_\infty(\{0,1\}^{B_{\text{in}}})} \leq \delta.$$

Consequently, if $\delta < 1/2$, thresholding p_θ at $1/2$ recovers the full truth table of f.

Theorem 2 and Corollary 1 are direct analogues of classical universal approximation theorems for MLPs for continuous function approximation, e.g. Funahashi (1989); Cybenko (1989); Hornik et al. (1989); Pinkus (1999); Kidger & Lyons (2020). They provide an existence guarantee independent of the "hardness" of representing the ground-truth function, both at the level of sampled circuits and at the level of the induced predictive marginal. Naturally, one wonders whether favourable rates are achievable; we now confirm this for sparse Boolean functions.

4.3 Efficient Reasoning

Indeed, analogously to quantitative universal approximation results for classical deep learning models, explicit approximation rates, while optimal in the worst-case cf. Yarotsky (2018), are often not the most compelling. Rather, the central question is whether the size of the resulting circuit representation can scale favourably with the intrinsic degrees of freedom of the problem. In classical neural network approximation theory, such favourable scaling is typically obtained under structural assumptions on the target function, notably some form of smoothness Yarotsky (2017); Suzuki (2019); Lu et al. (2021); Gühring & Raslan (2021); Siegel (2023). In our Boolean setting, an analogous notion of "nice structure" is sparsity, formalized by the target function being an S-junta with $S \ll B$; i.e. the target function only depends on S input bits out of the total B. We now show that, when the sparsity parameter S scales logarithmically with the number of inputs B, the number of neurons required by the Stochastic Boolean Circuit in Theorem 2 is polynomial; and in cases even linear, up to logarithmic factors.

Theorem 3 (Efficient Boolean Reasoning for $\mathcal{O}(\log(B))$-Juntas)**.** *Let $B, S \in \mathbb{N}_+$, $C > 0$, with $S \leq C \log(B)$ and let $f : \{0,1\}^B \to \{0,1\}$ be an S-junta. For every $0 < \delta \leq 1$, there is an $\hat{\boldsymbol{f}} : \Omega \times \{0,1\}^B \to \{0,1\}$ of depth Δ, width Υ, and at most N neurons satisfying* (9)*; where*

$$\Delta \in \mathcal{O}\big(\log(B) + \log(\log(B))\big), \qquad \Upsilon, N, B^\uparrow \in \mathcal{O}\big(B^C \log(B)\big)$$

where $C > 0$. In particular, if $0 < C \leq 1$, then $\Upsilon, N, B^\uparrow \in \mathcal{O}\big(B \log(B)\big)$.

5 Experimental Validation

We empirically study whether the induced probabilistic SBC predictor can be optimized for task-wise truth-table realization while retaining circuit-level Boolean semantics, unlike unconstrained MLPs.

Task-wise truth-table realization. Our main benchmark contains 2000 Boolean-function tasks with ambient input dimension $B \in \{2, 4, 6, \ldots, 20\}$, with 200 formulas for each value of B. Each task is

specified by a syntactic Boolean formula sampled from a common expression generator and evaluated on its full truth table. Thus, for each task, the training data are all assignments $x \in \{0,1\}^B$ together with labels $f(x)$. This is a task-wise realization setting rather than an across-task or foundation-model setup, and we do not use row-wise train/test splits over truth tables.

Relaxed prediction and evaluation. For an SBC parameter setting θ, the implementation returns a differentiable relaxed score $\hat{p}_\theta(x) \in [0,1]$ on each truth-table row. We threshold at $1/2$ and evaluate full truth-table exact match:

$$\hat{f}_\theta(x) \stackrel{\text{def}}{=} \mathbf{1}\{\hat{p}_\theta(x) \geq 1/2\}, \qquad \mathrm{EM}(\theta; f) \stackrel{\text{def}}{=} \mathbf{1}\left[\hat{f}_\theta(x) = f(x)\ \forall x \in \{0,1\}^B\right].$$

For SBC, this evaluates the relaxed score; for MLPs, the same criterion is applied to the thresholded real-valued predictor. Full experimental details are provided in Appendix F.

Predictive performance. Table 1 reports truth-table EM across seeds. We compare one canonical SBC predictor per task against ReLU MLPs under two matching rules: `neuron` matches MLP width/depth to the task-wise SBC architecture, while `param` matches trainable parameter budget. We report both the full benchmark and the large-truth-table slice $B \geq 18$; per-B results are in Appendix G.2. SBC outperforms the neuron-matched MLP in both blocks and is comparable to the parameter-matched MLP on the large slice.

Neuronwise Booleanity. Predictive accuracy alone does not distinguish a probabilistic circuit model from an unconstrained real-valued network. We therefore measure *Boolean Neuronal Representability* (BNR; Definition 1): a unit is BNR if it takes at most two values on the Boolean cube and can be recoded into $\{0,1\}$. For SBC, BNR is support-level: every sampled execution is Boolean-valued, although the predictive marginal is real-valued. For ReLU MLPs, we measure BNR post hoc over the truth table; Appendix G.5 shows that their primitive recoveries are biased toward constant-output gates, especially `TRUE`; therefore, BNR likely overstates meaningful non-constant Boolean structure.

Table 2: $\mathrm{BNR}_{\mathrm{exact}}$: fraction of units taking at most two truth-table values (mean $\pm$ std across seeds); "all hidden" averages hidden layers.

	SBC	MLP neuron	MLP param
First layer	1.000	0.114 ± 0.003	0.107 ± 0.002
All hidden	1.000	0.209 ± 0.002	0.199 ± 0.001

6 Conclusion

We proposed the Stochastic Boolean Circuit (SBC), a trainable distribution over circuits composed of binary, single-output Boolean gates. Every parameter setting yields almost surely valid Boolean-circuit executions (Theorem 1); with suitable parameters, sampled circuits compute any Boolean function with arbitrarily high probability (Theorem 2). For $\mathcal{O}(\log B)$-juntas, this can be achieved with near-linear scaling in the ambient dimension, up to logarithmic factors (Theorem 3). Empirically, we studied the induced probabilistic predictor on task-wise truth-table realization. The results highlight a predictive–structural tradeoff: SBCs provide probabilistic circuit-structured predictors with Boolean-valued sampled executions, while matched ReLU MLPs provide unconstrained real-valued predictors whose hidden units typically fail to be two-valued.

Limitations and Future Work. Our empirical study is task-wise and full-truth-table, so it does not address partial-truth-table generalization or across-task meta-generalization. The theory concerns the proof-level random-circuit model, whereas the implementation optimizes a differentiable probabilistic predictor; we do not prove that gradient descent finds optimal parameters. A further limitation is deterministic circuit extraction: local MAP decoding of a single circuit from the learned distribution can lose performance, reflecting a common difficulty in discrete structure learning, program synthesis, planning, and combinatorial optimization. Appendix G.3 reports preliminary diagnostics. Finally, we focus on circuits composed of binary, single-output Boolean gates and controlled truth-table realization benchmarks. Future work will study richer circuit and program classes, noisy or partial observations, across-task inference, and improved training or decoding methods that better concentrate probability mass on accurate discrete circuits.

AI USE STATEMENT

Large language models were used to assist with iterative editing, phrasing, and polishing of author-written manuscript text. The authors were responsible for the manuscript's content and reviewed all suggested revisions.

REPRODUCIBILITY STATEMENT

We provide an anonymized code archive containing the SBC implementation, baseline runners, fixed datasets, configurations, and commands for training and evaluation. Experimental details are given in Appendix F, and proofs are provided in Appendix E.

REFERENCES

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

Martin Anthony and Peter L Bartlett. *Neural network learning: Theoretical foundations*. cambridge university press, 2009.

Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, 2022.

Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? . In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, pp. 610–623, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383097. doi: 10.1145/3442188.3445922. URL https://doi.org/10.1145/3442188.3445922.

Lukas Berglund, Meg Tong, Maximilian Kaufmann, Mikita Balesni, Asa Stickland, Tomek Korbak, and Owain Evans. The reversal curse: Llms trained on "a is b" fail to learn "b is a". In *International Conference on Learning Representations*, volume 2024, pp. 18623–18642, 2024.

Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

Olivier Bournez, Johanne Cohen, and Adrian Wurm. A universal uniform approximation theorem for neural networks. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science (MFCS)*, Palaiseau, France, 2025.

Simon Bührer, Andreas Plesner, Till Aczel, and Roger Wattenhofer. Recurrent deep differentiable logic gate networks. In *Proceedings of the 2nd International Workshop on Edge and Mobile Foundation Models*, pp. 31–36, 2025.

Harsh Chaudhari, Ethan Rathbun, Hanna Foerster, Jamie Hayes, Matthew Jagielski, Milad Nasr, Ilia Shumailov, and Alina Oprea. Exploiting reasoning patterns in language models for indirect targeted poisoning, 2026. URL https://openreview.net/forum?id=hSLopCTOtT.

Justin Chih-Yao Chen, Archiki Prasad, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. Magicore: Multi-agent, iterative, coarse-to-fine refinement for reasoning. In *Proceedings of EMNLP*, 2025. URL https://aclanthology.org/2025.emnlp-main.1660/.

David Chiang. Transformers in uniform TC^0. *arXiv preprint*, arXiv:2409.13629, 2024.

Y Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. *UCLA. URL: http://starai. cs. ucla. edu/papers/ProbCirc20. pdf*, pp. 6, 2020.

Stephen Chung and Hava Siegelmann. Turing completeness of bounded-precision recurrent neural networks. *Advances in neural information processing systems*, 34:28431–28441, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. In *arXiv preprint arXiv:2110.14168*, 2021.

Stephen A. Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Perspectives in Logic. Cambridge University Press, Cambridge, 2010. ISBN 9780521517225.

Mark Craven and Jude Shavlik. Extracting tree-structured representations of trained networks. In D. Touretzky, M.C. Mozer, and M. Hasselmo (eds.), *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1995. URL https://proceedings.neurips.cc/paper_files/paper/1995/file/45f31d16b1058d586fc3be7207b58053-Paper.pdf.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8493–8502, Dublin, Ireland, 2022. Association for Computational Linguistics.

Kenji Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989. doi: 10.1016/0893-6080(89)90003-8.

Ingo Gühring and Mones Raslan. Approximation rates for neural networks with encodable weights in smoothness spaces. *Neural Networks*, 134:107–130, 2021.

Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 6–20. ACM, 1986.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *arXiv preprint arXiv:2103.03874*, 2021.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, Berlin, Heidelberg, 2012. ISBN 978-3-642-24507-5.

Marek Karpinski and Angus Macintyre. Polynomial bounds for vc dimension of sigmoidal and general pfaffian neural networks. *Journal of Computer and System Sciences*, 54(1):169–176, 1997.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergey Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tomas Tihon, et al. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://arxiv.org/abs/1912.09713.

Patrick Kidger and Terry Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pp. 2306–2327. PMLR, 2020.

Anastasis Kratsios, Dennis Zvigelsky, and Bradd Hart. Quantifying the limits of ai reasoning: Systematic neural network representations of algorithms. *arXiv preprint arXiv:2508.18526*, 2025.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 1831–1847, 2024.

Daniel Lowd and Pedro Domingos. Learning arithmetic circuits. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 383–392, 2008.

Jianfeng Lu, Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network approximation for smooth functions. *SIAM Journal on Mathematical Analysis*, 53(5):5465–5506, 2021.

Wolfgang Maass, Georg Schnitger, and Eduardo D. Sontag. On the computational power of sigmoid versus boolean threshold circuits. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 767–776, 1991.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Ankur Arjun Mali, Alexander G Ororbia II, and C Lee Giles. A neural state pushdown automata. *IEEE Transactions on Artificial Intelligence*, 1(3):193–205, 2021.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018.

Susan Margulies and Jason Morton. Polynomial-time solvable# csp problems via algebraic models and pfaffian circuits. *Journal of Symbolic Computation*, 74:152–180, 2016.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5:115–133, 1943. doi: 10.1007/BF02478259.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 2023.

William Merrill, Ashish Sabharwal, and Noah A. Smith. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856, 2022. doi: 10.1162/tacl_a_00493.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022. URL https://arxiv.org/abs/2209.11895.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Deep differentiable logic gate networks. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=vF3WefcoePW.

Allan Pinkus. *Approximation Theory of the MLP Model in Neural Networks*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 1999.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 689–690. IEEE, 2011.

Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyGBdo0qFm.

Meghana Arakkal Rajeev, Rajkumar Ramamurthy, Prapti Trivedi, Vikas Yadav, Oluwanifemi Bamgbose, Sathwik Tejaswi Madhusudhan, James Zou, and Nazneen Rajani. Cats confuse reasoning LLM: Query agnostic adversarial triggers for reasoning models. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=VrEPiN5WhM.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2302.04761.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: solving ai tasks with chatgpt and its friends in hugging face. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Noah Shinn, Beck Labash, Ashwin Gopinath, and Karthik R. Narasimhan. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2303.11366.

Parshin Shojaee, Seyed Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=YghiOusmvw.

Jonathan W. Siegel. Optimal approximation rates for deep relu neural networks on sobolev and besov spaces. *Journal of Machine Learning Research*, 24(357):1–52, 2023. URL http://jmlr.org/papers/v24/23-0025.html.

Hava T Siegelmann. *Neural networks and analog computation: beyond the Turing limit*. Springer Science & Business Media, 2012.

Kai-Yeung Siu, Jehoshua Bruck, Thomas Kailath, and Thomas Hofmeister. Depth efficient neural networks for division and related problems. *IEEE Transactions on Information Theory*, 39(3): 946–956, 1993. doi: 10.1109/18.256501.

Peiyang Song, Pengrui Han, and Noah Goodman. A survey on large language model reasoning failures. In *2nd AI for Math Workshop @ ICML 2025*, 2025a. URL https://openreview.net/forum?id=hsgMn4KBFG.

Zhao Song, Song Yue, and Jiahao Zhang. Thinking isn't an illusion: Overcoming the limitations of reasoning models via tool augmentations. *arXiv preprint arXiv:2507.17699*, 2025b.

Taiji Suzuki. Adaptivity of deep reLU network for learning in besov and mixed smooth besov spaces: optimal rate and curse of dimensionality. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1ebTsActm.

Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7), 2021.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2201.11903.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. In *Proceedings of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a. URL https://arxiv.org/abs/2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b. URL https://arxiv.org/abs/2210.03629.

Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94: 103–114, 2017.

Dmitry Yarotsky. Optimal approximation of continuous functions by very deep relu networks. In *Conference on learning theory*, pp. 639–649. PMLR, 2018.

Eric Zelikman, Yuhuai Wu, and Noah D Goodman. Star: Self-taught reasoner. In *Proceedings of the NIPS*, volume 22, 2022.

Jun Zhao, Jingqi Tong, Yurong Mou, Ming Zhang, Qi Zhang, and Xuan-Jing Huang. Exploring the compositional deficiency of large language models in mathematical reasoning through trap problems. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 16361–16376, 2024.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. {PoisonedRAG}: Knowledge corruption attacks to {Retrieval-Augmented} generation of large language models. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 3827–3844, 2025.

Roadmap of the appendix. The appendix collects technical details, proofs, and additional experimental material omitted from the main text. Appendix A records elementary facts about binary, single-output Boolean gates, including the enumeration of all 16 such gates and the one-gate NOT example illustrating why unconstrained ReLU parameterizations need not easily realize Boolean gates exactly. Appendix B develops additional properties of Boolean Neuronal Representability (BNR), including examples and supporting results used to justify the neuronwise Booleanity diagnostics in the experiments. Appendix E contains the complete proofs of the main theoretical results and the auxiliary propositions used in the model construction. When a longer proof is collected there rather than placed immediately after its statement, an explicit proof-location line below the statement gives the relevant page. Appendix F gives the full experimental protocol, including dataset generation, model configurations, matching rules for the MLP baselines, training details, decoding procedures, and the numerical implementation of the BNR diagnostics. Finally, Appendix G reports additional experimental, including exact-match and BNR summaries, gate-recoverability probes, and gate-type histograms.

A Elementary Properties of Boolean Gates

A.1 Table of all 16 Binary, Single-Output Boolean Gates

Table 3 lists all binary, single-output Boolean gates. The term *single-output* means that each gate computes one Boolean value; that value may be reused by arbitrarily many downstream gates, so the graph-theoretic fan-out is unrestricted.

			$Z^{(i)}$			
Gate number	Gate	Expression	(0,0)	(0,1)	(1,0)	(1,1)
g_1	Constant FALSE	0	0	0	0	0
g_2	AND	$A \wedge B$	0	0	0	1
g_3	A AND NOT B	$A \wedge \neg B$	0	0	1	0
g_4	PROJ$_A$	A	0	0	1	1
g_5	NOT A AND B	$\neg A \wedge B$	0	1	0	0
g_6	PROJ$_B$	B	0	1	0	1
g_7	XOR	$A \oplus B$	0	1	1	0
g_8	OR	$A \vee B$	0	1	1	1
g_9	NOR	$\neg(A \vee B)$	1	0	0	0
g_{10}	XNOR	$A \leftrightarrow B$	1	0	0	1
g_{11}	NOT B	$\neg B$	1	0	1	0
g_{12}	A OR NOT B	$A \vee \neg B$	1	0	1	1
g_{13}	NOT A	$\neg A$	1	1	0	0
g_{14}	NOT A OR B	$\neg A \vee B$	1	1	0	1
g_{15}	NAND	$\neg(A \wedge B)$	1	1	1	0
g_{16}	Constant TRUE	1	1	1	1	1

Table 3: Enumeration of all elementary Boolean gates; we denote the set $\mathbb{G}^\star \stackrel{\text{def}}{=} \{g_i\}_{i=1}^{16}$.

A.2 Boolean and Binary Logical Gates

Note that there are two 0-ary binary gates; namely the constant TRUE $\stackrel{\text{def}}{=} 1$ or FALSE $\stackrel{\text{def}}{=} 0$ gates. Furthermore, there can only be two "truly" unary gates namely the identity gate sending any $x \in \{0,1\}$ to itself and the negation gate NOT sending x to $1 + x \mod (2)$. Our first lemma observes that every 0-ary and each unary gate can be realized as one of the binary gates in $\mathbb{G}^\star$ upon composing with the projection $\pi : \{0,1\}^2 \ni (x_1, x_2) \to x_1 \in \{0,1\}$.

Lemma 1 (Embedding of Unary Gates)**.** *There exist two constant binary gates; namely g_1 and g_{16} in Table 3. Moreover, let $g : \{0,1\} \to \{0,1\}$ be a unary Boolean gate. Then, there exists a binary Boolean gate $g^\uparrow$ such that $g^\uparrow = g \circ \pi$. Consequently, we may study only binary Boolean gates and subsume our study of constant (0-ary) and unary Boolean gates.*

Proof of Lemma 1. By definition, the gates g_1 and g_{16} in Table 3 are constant. We also have that $g_4 = \mathrm{ID} \circ \pi$ and $g_{13} = \mathrm{NOT} \circ \pi$; the unary constant true and false gates are also given by g_{16} and g_1 by composition with π; respectively. □

We denote the set of all standard Boolean gates $\mathbb{G}^{\text{Bool}} \stackrel{\text{def}}{=} \{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\}$ as a subset of $\mathbb{G}^\star$, however as the next result shows; there exists exactly 13 other possible binary Boolean gates not included in $\mathbb{G}^{\text{Bool}}$ and these are listed in Table 3.

Lemma 2 (Enumeration of $\mathbb{G}^\star$)**.** *There exists exactly* 16*,* 2*-ary binary Boolean gates. Moreover, these are enumerated in Table 3.*

Proof of Lemma 2. Every Boolean gate is a map $g : \{0,1\}^2 \to \{0,1\}$. Since $\#\{0,1\} = 2$, then $\#\{0,1\}^2 = 4$; whence the cardinality of $\{0,1\}^{\{0,1\}^2}$ is $2^4 = 16$. For the second statement, note that each gate in Table 3 has a distinct truth assignment. □

A.3 A One-Gate Illustration: Learning NOT

We record a simple calculation showing that even realizing a single Boolean gate with an unconstrained ReLU neuron requires satisfying a low-dimensional and non-convex set of parameter constraints.

Consider the one-dimensional ReLU model

$$f_\theta : \mathbb{R} \to \mathbb{R}, \qquad f_\theta(x) := \theta_3 \,\mathrm{ReLU}(\theta_1 x + \theta_2), \qquad \theta = (\theta_1, \theta_2, \theta_3) \in \mathbb{R}^3.$$

We ask when f_θ computes the Boolean gate NOT on $\{0,1\}$, namely when

$$f_\theta(0) = 1, \qquad f_\theta(1) = 0.$$

Lemma 3. *The model $f_\theta(x) = \theta_3 \,\mathrm{ReLU}(\theta_1 x + \theta_2)$ computes* NOT *on $\{0,1\}$ if and only if*

$$\theta_2 > 0, \qquad \theta_3 = \frac{1}{\theta_2}, \qquad \theta_1 + \theta_2 \le 0.$$

Consequently, the set of parameters realizing NOT *is a low-dimensional, non-convex subset of $\mathbb{R}^3$.*

Proof. First, the condition $f_\theta(0) = 1$ gives $\theta_3 \,\mathrm{ReLU}(\theta_2) = 1$. In particular, $\mathrm{ReLU}(\theta_2) > 0$, and therefore $\theta_2 > 0$. Hence $\mathrm{ReLU}(\theta_2) = \theta_2$, so $\theta_3\theta_2 = 1$, which implies $\theta_3 = \frac{1}{\theta_2}$.

Next, the condition $f_\theta(1) = 0$ gives $\theta_3 \,\mathrm{ReLU}(\theta_1 + \theta_2) = 0$. Since $\theta_3 = 1/\theta_2$ and $\theta_2 > 0$, we have $\theta_3 \neq 0$. Therefore, $\mathrm{ReLU}(\theta_1 + \theta_2) = 0$, which is equivalent to $\theta_1 + \theta_2 \le 0$. This proves the claimed characterization.

The parameter set is contained in the surface

$$\theta_3 = \frac{1}{\theta_2}, \qquad \theta_2 > 0,$$

together with the inequality $\theta_1 + \theta_2 \le 0$. Thus it has lower dimension than the ambient space $\mathbb{R}^3$. It is also non-convex. For example,

$$\theta = (-1, 1, 1), \qquad \theta' = (-2, 2, 1/2)$$

both satisfy the above constraints. However their midpoint is

$$\bar{\theta} = \frac{\theta + \theta'}{2} = \left(-\frac{3}{2}, \frac{3}{2}, \frac{3}{4}\right),$$

and this does not satisfy $\bar{\theta}_3 = \frac{1}{\bar{\theta}_2}$.

□

B BOOLEAN NEURONAL REPRESENTABILITY (BNR)

We now include a few more details on the BNR property of real-valued functions.

Example 1. Let $a > 0$, the map $f : \{0,1\}^2 \ni x \to a \cdot x \in \mathbb{R}^2$ is BNR.

Example 2. The map $f : \{0,1\}^2 \ni x \to (x_1 + x_2, x_1) \in \mathbb{R}^2$ is not BNR since $f((0,0))_1 = 0$, $f((0,1))_1 = 1$, and $f((1,1))_1 = 2$; meaning it is impossible to assign two distinct truth values to these three distinct values.

Example 3 (ReLU neurons need not be BNR)**.** Let $2 \leq B \in \mathbb{N}_+$. Let $0 < a_1 < a_2$ and define $a \stackrel{\text{def}}{=} (a_i)_{i=1}^B$. Thus $\mathrm{ReLU}(a^\top 0) = 0 < \mathrm{ReLU}(a^\top e_1) = a_1 < \mathrm{ReLU}(a^\top e_2) = a_2$ so $f(\cdot) \stackrel{\text{def}}{=} \mathrm{ReLU}(a^\top \cdot + 0)$ fails Proposition 3 (ii); whence f is not BNR.

Proposition 1 (Random ReLU neurons fail to be BNR w.h.p.)**.** *Let $3 \leq B \in \mathbb{N}_+$ and let $a \stackrel{\text{def}}{=} (a_i)_{i=1}^B \in \mathbb{R}^B$ have i.i.d. coordinates, where each a_i is drawn from an atom-free distribution symmetric about 0. Define $f(\cdot) \stackrel{\text{def}}{=} \mathrm{ReLU}(a^\top \cdot + 0)$. Then, with probability at least $1 - \frac{B+2}{2^B}$, there exist* distinct *indices $i, j, k \in [B]_+$ such that*

$$\mathrm{ReLU}(a^\top e_i) = 0 < \mathrm{ReLU}(a^\top e_j) = a_j < \mathrm{ReLU}(a^\top e_k) = a_k,$$

and hence $f(\cdot)$ fails Proposition 3 (ii); whence f is not BNR.

Proof of Proposition 1. Since each a_i is drawn i.i.d. from an atom-free distribution symmetric about 0, we have

$$\mathbb{P}(a_i > 0) = \mathbb{P}(a_i < 0) = \tfrac{1}{2}, \qquad \mathbb{P}(a_i = 0) = 0,$$

and moreover, by atom-freeness, $\mathbb{P}(\exists\, i \neq j :\ a_i = a_j) = 0$, so that, almost surely, the values $a_1, \ldots, a_B$ are pairwise distinct.

We have $a^\top e_i = a_i$ and hence $\mathrm{ReLU}(a^\top e_i) = \max\{a_i, 0\}$. Let $N_+ \stackrel{\text{def}}{=} |\{i \in [B]_+ : a_i > 0\}|$, so $N_+ \sim \mathrm{Binomial}(B, \frac{1}{2})$. If $2 \leq N_+ \leq B - 1$, then there exist distinct $i, j, k \in [B]_+$ with $a_i < 0 < a_j < a_k$ (note that $a_1, \ldots, a_B$ are a.s. distinct), and therefore

$$\mathrm{ReLU}(a^\top e_i) = 0 < \mathrm{ReLU}(a^\top e_j) = a_j < \mathrm{ReLU}(a^\top e_k) = a_k.$$

Hence

$$\mathbb{P}\big(\exists\, i, j, k \text{ distinct s.t. } 0 = \mathrm{ReLU}(a^\top e_i) < \mathrm{ReLU}(a^\top e_j) < \mathrm{ReLU}(a^\top e_k)\big) \ \geq\ \mathbb{P}(2 \leq N_+ \leq B{-}1).$$

Finally,

$$\mathbb{P}(2 \leq N_+ \leq B-1) = 1 - \mathbb{P}(N_+ = 0) - \mathbb{P}(N_+ = 1) - \mathbb{P}(N_+ = B) = 1 - \frac{B+2}{2^B}.$$

Thus, with probability at least $1 - \frac{B+2}{2^B}$, the above inequalities hold. Thus, $f(\cdot)$ fails Proposition 3 (ii), and hence $f(\cdot)$ is not BNR. □

The following provides a simple characterization of the BNR property which can be used as an efficient criterion for verifying if a neuron is Boolean neuronally representable (property (ii)) as canonical choice of a BNR.

Proposition 2 (Characterization of BNR - Extended Version of Proposition 3)**.** *Let $B_{\text{in}}, B_{\text{out}} \in \mathbb{N}_+$ and $f : \{0,1\}^{B_{\text{in}}} \to \mathbb{R}^{B_{\text{out}}}$. Then, the following are equivalent:*

(i) f is BNR,

(ii) For every $i \in [B_{\text{out}}]_+$, $\#\{f(x)_i : x \in \{0,1\}^{B_{\text{in}}}\} \leq 2$,

(iii) The map $\mathcal{R}_f : \mathbb{R}^{B_{\text{out}}} \to \{0,1\}^{B_{\text{out}}}$ sending any $x \in \mathbb{R}^{B_{\text{out}}}$ to

$$\mathcal{R}_f(x) \stackrel{\text{def}}{=} \big(I(x_i = f(\mathbf{0})_i)\big)_{i=1}^{B_{\text{out}}},$$

where $\mathbf{0} \stackrel{\text{def}}{=} (0 \ldots 0) \in \{0,1\}^{B_{\text{in}}}$, is a Boolean neuronal representation for f.

We refer to $\mathcal{R}_f$ as the canonical BNR, whenever f is BNR.

Proof location. See page 18.

Definition 3 (BNR Density)**.** Let $L \in \mathbb{N}_+$ and let $B_1, \ldots, B_{L+1} \in \mathbb{N}_+$. For each $l \in [L]_+$ let $f_l : \mathbb{R}^{B_l} \to \mathbb{R}^{B_{l+1}}$ and consider the composite map $f \stackrel{\text{def}}{=} f_L \circ \cdots \circ f_1$. Define the sets $\mathcal{D}_0^f \stackrel{\text{def}}{=} \{0,1\}^{B_1}$ and, for every $l \in [L]_+$, iteratively define

$$\mathcal{D}_l^f \stackrel{\text{def}}{=} f_l(\mathcal{D}_{l-1}^f).$$

The BNR density of f is $\mathrm{BNR}(f) \stackrel{\text{def}}{=} \sum_{l=1}^{L} \sum_{i=1}^{B_{l+1}} \frac{I\left(\#\{(f_l(x))_i : x \in \mathcal{D}_{l-1}^f\} \leq 2\right)}{LB_{l+1}}$.

Example 4 (Boolean Functions (and Gates) Have BNR Density 1)**.** Let $L = 1$, $B_1 = B_{\text{in}}$, and $B_2 = B_{\text{out}}$. For every $f : \{0,1\}^{B_{\text{in}}} \to \{0,1\}^{B_{\text{out}}}$ we have $\mathrm{BNR}(f) = 1$.

B.1 Proofs Surrounding BNR

Proof of Proposition 2. Assume (i) and for every $i \in [B_{\text{out}}]_+$ write

$$S_i \stackrel{\text{def}}{=} \{f(x)_i : x \in \{0,1\}^{B_{\text{in}}}\}.$$

If there exists some $i \in [B_{\text{out}}]_+$ such that $\#S_i \geq 3$, then the map $\mathcal{R}_i : S_i \to \{0,1\}$ cannot be injective by the pigeonhole principle, which contradicts the definition of BNR. This proves (ii).

Next, assume (ii). Fix $i \in [B_{\text{out}}]_+$. Since $\#S_i \leq 2$, there is at most one element of S_i different from $f(\mathbf{0})_i$. Hence the predicate $x_i = f(\mathbf{0})_i$ separates the elements of S_i, implying that $(\mathcal{R}_f)_i\big|_{S_i}$ is injective. Since i was arbitrary, $\mathcal{R}_f$ is a Boolean neuronal representation for f, proving (iii).

Finally, the implication (iii)⇒(i) is immediate. □

Example 5. The map $f : \{0,1\}^2 \ni x \to (x_1 + x_2, x_1) \in \mathbb{R}^2$ is not BNR since $f((0,0))_1 = 0$, $f((0,1))_1 = 1$, and $f((1,1))_1 = 2$; meaning it is impossible to assign two distinct truth values to these three distinct values.

The following provides a simple characterization of the BNR property which can be used as an efficient criterion for verifying if a neuron is Boolean neuronally representable (property (ii)) as canonical choice of a BNR.

Proposition 3 (Characterization of BNR)**.** *Let $B_{\text{in}}, B_{\text{out}} \in \mathbb{N}_+$ and $f : \{0,1\}^{B_{\text{in}}} \to \mathbb{R}^{B_{\text{out}}}$. Then, the following are equivalent:*

(i) f is BNR,

(ii) For every $i \in [B_{\text{out}}]_+$, $\#\{f(x)_i : x \in \{0,1\}^{B_{\text{in}}}\} \leq 2$.

Proof. This is exactly the equivalence of (i) and (ii) in Proposition 2. □

C Auxiliary Properties of SBC Components

C.1 Explaining Each Component of the SBC Model

We now justify our parameterizations of each component of the proposed Stochastic Boolean Circuit model by briefly summarizing their key interpretability and expressivity properties.

For any choice of weight matrix W, the stochastic bit-lifting channel $\mathcal{L}(\cdot \,|\, W)$ always defines a random function taking values in the $B^\uparrow$-dimensional Boolean cube. Moreover, W can be chosen, during training, so that any desired subset of variables in $\{0,1\}^B$, or their negations, is repeated or omitted an arbitrary number of times.

Proposition 4 (Stochastic Bit-Lifting Channels)**.** *Let $B, B^\uparrow \in \mathbb{N}_+$, let $W \in \mathbb{R}^{B^\uparrow \times 2B}$, and let $\mathbf{X}^{\uparrow:W}$ be defined in* (3)*. Then, $\mathbf{X}^{\uparrow:W}(x) \in \{0,1\}^{B^\uparrow}$ $\mathbb{P}$-a.s.*
Moreover, if W is binary with only one non-zero entry per row then: for every $0 < \delta \leq 1$ there exists some $\eta > 0$ such that

$$\mathbb{P}\left(\left(\forall i \in [B]_+\right) \#\left\{X_j^{\uparrow:\eta W} = x_i\right\}_{j \in [B^\uparrow]_+} = \|W_{:i}\|_0 \text{ and } \#\left\{X_j^{\uparrow:\eta W} = \neg\, x_i\right\}_{j \in [B^\uparrow]_+} = \|W_{:i+B}\|_0 \right) \geq 1 - \delta.$$

Proof location. See page 21.

Next, we show that each neuron in SBC selects exactly one pair of incoming edges from the previous layer and computes exactly one of the 16 binary, single-output Boolean gates. The selected output may be consumed by multiple downstream neurons.

Proposition 5 (Only, and Any, Two Edges Can Be Sampled)**.** *Let $B \in \mathbb{N}_+$. For all $\eta > 0$,*

(i) ***Non-Singular Probabilistic Interpretability:*** *The range of S_η is $\dot{\Delta}_B \times \dot{\Delta}_B$,*

(ii) ***Smoothness:*** *The map S_η is (C^∞-)smooth.*

Furthermore, for every $0 < \varepsilon < \frac{1}{2}$, there exists a temperature threshold $\eta_\varepsilon > 0$ such that: for each $i \in [B]_+$,

(iii) ***No Doubled Edges:*** $\sup_{\substack{W_2 \in \mathbb{R}^B \\ \exists\, j \neq i:\ (W_2)_j > (W_2)_i}} \mathrm{S}_{\eta_\varepsilon}\left(e_i, W_2\right)_{i:2} \in [0, \varepsilon].$

(iv) ***Recovery****: If $j \in [B]_+$ and $j \neq i$ then $\left\| \mathrm{S}_{\eta_\varepsilon}\left(e_i, e_j\right) - (e_i, e_j)\right\|_1 < \varepsilon$.*

In particular, if $(\mathbf{E}_{i:1}, \mathbf{E}_{i:2}) \sim S_{\eta_\varepsilon}(e_i, e_j)$ then $\mathbb{P}\big((\mathbf{E}_{i:1}, \mathbf{E}_{i:2}) = (e_i, e_j)\big) \geq 1 - \varepsilon$.

Proof location. See page 21.

We show that stochastic Boolean gates are meaningful; i.e., they are in $\mathbb{G}^\star$, and that we can set a stochastic Boolean gate to be a *specific* gate in $\mathbb{G}^\star$ with arbitrarily high probability.

Proposition 6 (Stochastic Boolean Gates: Sample Boolean Gates)**.**
For every $W \in \mathbb{R}^{16}$, $\boldsymbol{\sigma}^W|_{\{0,1\}^2} \in \mathbb{G}^\star$.

Proof location. See page 20.

Proposition 7 (Stochastic Boolean Gates: Universally Sample Boolean Gates)**.** *For every failure probability $\delta \in (0, 1]$ and each $i \in [16]_+$, if $\eta \geq \ln\left(\frac{15}{\delta} - 15\right)$ then*

$$\mathrm{TV}\left(\mathrm{Law}(\boldsymbol{\sigma}^{\eta e_i}|_{\{0,1\}^2}), \delta_{g_i}\right) \leq \delta. \tag{10}$$

In particular, $\mathbb{P}\big(\boldsymbol{\sigma}^{\eta e_i}|_{\{0,1\}^2} = g_i\big) \geq 1 - \delta$.

Proof location. See page 20.

D Corner-Exact Gate Interpolants

D.1 Gates: Trainable Stochastic Boolean Gates

Let $\bar{x}_1, \ldots, \bar{x}_4$ be the elements of $\{0,1\}^2$ ordered as $(0,0), (0,1), (1,0), (1,1)$. For every truth assignment $Z \in \{0,1\}^4$, define the quadratic interpolant (note differentiable) $\tilde{\sigma} : \{0,1\}^2 \to \{0,1\}$ for each $x \in \{0,1\}^2$ by

$$\tilde{\sigma}(x|Z) \stackrel{\text{def}}{=} \sum_{i=1}^{4} \varsigma(x - \bar{x}_i) Z_i, \quad \text{where} \quad \varsigma(x) \stackrel{\text{def}}{=} \begin{cases} c\, e^{1/(4\|x\|_2^2 - 1)} & \text{if } \|x\|_2 < \frac{1}{2} \\ 0 & \text{if } \|x\|_2 \geq \frac{1}{2} \end{cases} \tag{11}$$

where $c \stackrel{\text{def}}{=} e$. Stacking this, we extend (11) to $\bar{\sigma} : \{0,1\}^2 \to \{0,1\}^{16}$ sending any $\{0,1\}^2$ to

$$\bar{\sigma}(x) \stackrel{\text{def}}{=} \big(\tilde{\sigma}(x|Z)\big)_{Z \in \{0,1\}^4}. \tag{12}$$

We now create an $\mathbb{R}^{16}$-valued non-linearity whose coordinates are simply $(\tilde{\sigma}(x|Z))$ with the Z running through the different values of the four dimensional Boolean hypercube. Namely, we consider the non-linearity $\sigma : \mathbb{R}^2 \to \mathbb{R}^{16}$ given for any $x \in \mathbb{R}^2$ by

$$\sigma(x) = \big(\tilde{\sigma}(x|Z^{(i)})\big)_{i=1}^{16}, \tag{13}$$

where $Z^{(1)} = (0,0,0,0), \ldots, Z^{(16)} = (1,1,1,1)$ is an enumeration of $\{0,1\}^4$.

Consider the stochastic Boolean gate selector defined for any trainable parameter $W_\Sigma \in \mathbb{R}^{16}$ and any input $x \in \mathbb{R}^2$ by $\Sigma(x \mid W_\Sigma) \stackrel{\text{def}}{=} \mathrm{SM}_{16}(W_\Sigma)^\top \sigma(x)$. The quantity is the pointwise mean, of any random variable $\boldsymbol{\sigma}^{W_\Sigma} : \Omega \to C^\infty(\mathbb{R}^2, \mathbb{R})$ given for any outcome $\omega \in \Omega$ by

$$\begin{aligned} \boldsymbol{\sigma}^{W_\Sigma}(\omega)(x) &\stackrel{\text{def}}{=} \sum_{i=1}^{16} X_i^{W_\Sigma}(\omega)\, \tilde{\sigma}(x|Z^{(i)}) \\ \text{s.t. } \mathbb{P}(X^{W_\Sigma} = e_i) &\stackrel{\text{def}}{=} \big(\mathrm{SM}_{16}(W_\Sigma)\big)_i \end{aligned} \tag{14}$$

for each $i \in [16]_+$, where x is an arbitrary element of $\mathbb{R}^2$.

E PROOFS

Our proof will follow a variant of the surgery approach of Kratsios et al. (2025). Namely, we first exhibit a worst-case Boolean circuit on the gates $\{\mathrm{AND}, \mathrm{OR}, \mathrm{PROJ}_B, \mathrm{NOT}\}$ which implement any given S-Junta. We then swap the gates, this is the surgery technique of Kratsios et al. (2025), for networks which perform the same computation but now with high probability. Together, this will give universal reasoning.

Proposition 8. (Smooth Compactly Supported Exhaustion of Elementary Boolean Gates) *For any $x \in \{0,1\}^2$ we have*

$$\sigma(x) = \big(g_i(x)\big)_{i=1}^{16}$$

where for each $i \in [16]_+$ g_i is the elementary Boolean gate in Table 3.
Additionally, σ is (C^∞-)smooth and compactly supported.

First, we confirm that any stochastic Boolean gate as in $\boldsymbol{\sigma}^W$ is almost surely a valid Boolean gate when sampled.

Proof of Proposition 8. Since ς is a composition of the polynomial function $x \mapsto \|2x\|^2$ with the smooth compactly supported bump function $x \mapsto I_{|x|<1} e^{-\frac{1}{|x|^2-1}}$ then it is smooth. Since $\tilde{\sigma}(\cdot|Z)$ is a linear combination of shifts of these maps so is each $\tilde{\sigma}(\cdot|Z)$ where $Z \in \{0,1\}^4$. Since σ is the direct sum of compactly supported smooth functions it is also smooth and compactly supported.

By construction, for each $\bar{x}_i \in \{0,1\}^2$ and every $x \in \{0,1\}^2$ we have $\varsigma(x - \bar{x}_i) = 1$ if and only if $x = \bar{x}_i$ and $\varsigma(x - \bar{x}_i) = 0$ otherwise. Consequently, for every $x \in \{0,1\}^2$ and each $i \in [4]_+$

$$\tilde{\sigma}(x|Z) = \sum_{i=1}^{4} \varsigma(x - \bar{x}_i) Z_i = \sum_{i=1}^{4} I_{x=\bar{x}_i} Z_i. \tag{15}$$

By the interpolation identity in (15) we have that: for each $i \in [4]_+$, $\tilde{\sigma}(x|Z) = Z_i$ if and only if $x = \bar{x}_i$. □

Proof of Proposition 6. For any $W \in \mathbb{R}^{16}$, by definition of $\boldsymbol{\sigma}^W$, we have that $\boldsymbol{\sigma}^W \in \{\tilde{\sigma}(\cdot|Z_i)\}_{i \in [16]_+}$. The result then follows from Proposition 8. □

In the following proof, we use $\mathcal{P}(\mathcal{X})$ to denote the space of Borel probability measures on a metric space $\mathcal{X}$. We metrize $\mathcal{P}(\mathcal{X})$ with the total variation metric TV, which assigns to any probability measures $\mathbb{P}, \mathbb{Q} \in \mathcal{P}(\mathcal{X})$

$$\mathrm{TV}(\mathbb{P}, \mathbb{Q}) \stackrel{\text{def}}{=} \sup_A |\mathbb{P}(A) - \mathbb{Q}(A)|$$

where the supremum is taken overall Borel subsets A of $\mathcal{X}$. We write $X \sim Y$ for two random variables, defined on the same probability space, whose laws coincide.

Proof of Proposition 7. Fix $\eta > 0$, to be specified retroactively. By Proposition 8, $\boldsymbol{\sigma}^W|_{\{0,1\}^2} \sim \mathbb{P}^W \stackrel{\text{def}}{=} \sum_{i=1}^{16} \mathrm{SM}(W)_i \delta_{g_i}$ where, $W = \eta e_i$. In particular, $\mathbb{P}^W \in \mathcal{P}\big(\{g_i\}_{i\in[16]_+}\big)$ equipped with the metric TV isometric to $(\Delta_{16}, \frac{1}{2}\|\cdot\|_1)$ with isometry $\varphi : \Delta_{16} \to \mathcal{P}(\mathbb{G}^\star)$ sending any $w \in \Delta_{16}$ to $\mathbb{P}_w \stackrel{\text{def}}{=} \sum_{i=1}^{16} w_i \, \delta_{g_i}$. consequently, and by definition of $\mathbb{P}^W$ we have, $\mathrm{TV}(\mathbb{P}^W, \delta_{g_i}) = \frac{1}{2}\|\mathrm{SM}(\eta e_i) - e_i\|_1$. Now, since $\|\mathrm{SM}(\eta e_i) - e_i\|_1 = \frac{2(G-1)}{e^\eta + (G-1)}$; where $G = 16$ then,

$$\mathrm{TV}(\mathbb{P}^{\eta e_i}, \delta_{g_i}) = \frac{1}{2}\|\mathrm{SM}(\eta e_i) - e_i\|_1 = \frac{15}{e^\eta + 15}. \tag{16}$$

Therefore, if $\eta \geq \ln\big(\frac{15}{\delta} - 15\big)$ then (16) implies that $\mathrm{TV}(\mathbb{P}^{\eta e_i}, \delta_{g_i}) \leq \delta$. By definition of the total variation distance, we have

$$\mathbb{P}\big(\boldsymbol{\sigma}^{\eta e_i}|_{\{0,1\}^2} = g_i\big) = 1 - \mathrm{TV}(\mathbb{P}^{\eta e_i}, \delta_{g_i}) = 1 - \frac{1}{2}\|\mathrm{SM}(\eta e_i) - e_i\|_1 \geq 1 - \delta$$

completing our proof. □

Example 6. Let $B = 2$ and $B^\uparrow = 3$. Fix any $x \in \{0,1\}^2$. For every $\eta > 0$, define $W^\eta \in \mathbb{R}^{B^\uparrow \times 2B}$ by $W^\eta = \eta \begin{pmatrix} e_1, & e_3, & e_2 \end{pmatrix}^\top$, where (e_1, e_2, e_3, e_4) denotes the standard basis of $\mathbb{R}^4$. Then, as $\eta \to \infty$, the random variables $(\mathbf{X}^{\uparrow:W^\eta})_{\eta>0}$ tends to the degenerate random variable taking value $(x_1, \neg x_1, x_2)$ $\mathbb{P}$-a.s.

Proof of Proposition 4. The first claim is trivial. For the second, the argument is nearly analogous to the proof of Proposition 7, mutatis mutandis. □

Proof of Proposition 5. Now observe that Property (i) holds since the range of SM is $\dot{\Delta}_B$.

Property (ii) holds since SM is analytic, as well as the Hadamard product, multiplication and addition by constants; as well as the composition and concatenation of (C^∞-)smooth functions.

To show (iii), we first prove two properties of SM. First we will show that SM is monotone. That is, given $W \in \mathbb{R}^B$ with $(W)_i < (W)_j$ for $i, j \in B$ and $\eta > 0$,

$$\mathrm{SM}(\eta W)_i < \mathrm{SM}(\eta W)_j. \tag{17}$$

This follows from the definition of SM and the monotonicity of e:

$$\mathrm{SM}(\eta W)_i = \frac{e^{\eta(W)_i}}{\sum_{k=1}^B e^{\eta(W)_k}} < \frac{e^{\eta(W)_j}}{\sum_{k=1}^B e^{\eta(W)_k}} = \mathrm{SM}(\eta W)_j. \tag{18}$$

Towards the second property, assume again that $(W)_i < (W)_j$. Then we will show that for all $\varepsilon > 0$, there exists some $\eta > 0$ such that

$$\mathrm{SM}(\eta W)_i < \epsilon. \tag{19}$$

Now it follows by the definition of SM that

$$\mathrm{SM}(\eta W)_i = \frac{e^{\eta(W)_i}}{\sum_{k=1}^B e^{\eta(W)_k}} < \frac{e^{\eta(W)_i}}{e^{\eta(W)_j}} = e^{-\eta((W)_j - (W)_i)}. \tag{20}$$

Setting $e^{-\eta((W)_j-(W)_i)} < \varepsilon$ yields

$$\eta > \frac{\ln(1/\varepsilon)}{(W)_j - (W)_i}. \tag{21}$$

Now towards (iii): Notice that the definition of SM and (17) necessitate that

$$\eta(1 - \mathrm{SM}(\eta e_i)_i)\,\mathrm{SM}(\eta(W_2))_i < \eta(1 - \mathrm{SM}(\eta e_i)_j)\,\mathrm{SM}(\eta(W_2))_j, \tag{22}$$

and so by (19), given $\varepsilon > 0$, there exists $\eta^{\star:1} > 0$ such that for all $\eta > \eta^{\star:1}$ the claim holds.

We now prove (iv). For every $k \in [B]_+ \setminus \{i\}$,

$$\lim_{\eta\to\infty} \mathrm{SM}(\eta e_i)_k = \lim_{\eta\to\infty} \frac{1}{(B-1)e^{\eta 0} + e^{\eta}} = \lim_{\eta\to\infty} \frac{1}{(B-1)+e^{\eta}} = 0. \tag{23}$$

Since the range of SM belongs to the B-simplex then

$$\lim_{\eta\to\infty} \mathrm{SM}(\eta e_i)_i = \lim_{\eta\to\infty} 1 - \sum_{k>1}^{B} \mathrm{SM}(\eta e_i)_k = \lim_{\eta\to\infty} 1 - 0 = 1. \tag{24}$$

Together (24) and (23) imply that there exists some $\eta^{\star:2} > 0$ such that: for all $\eta \geq \eta^{\star:2}$ we have

$$\|\mathrm{SM}(\eta e_i) - e_i\|_1 < \tfrac{\varepsilon}{2}. \tag{25}$$

Now take $w = e_j$ such that $j \neq i$. Simplifying $\mathrm{SM}\left(\eta\big((1-\mathrm{SM}(\eta e_i)) \odot \mathrm{SM}(\eta e_j)\big)\right)_j$ we find that it equals to

$$\frac{\exp\left(\eta \frac{e^{\eta}(e^{\eta}+B-2)}{(e^{\eta}+(B-1))^2}\right)}{\exp\left(\eta \frac{B-1}{(e^{\eta}+(B-1))^2}\right) + \exp\left(\eta \frac{e^{\eta}(e^{\eta}+B-2)}{(e^{\eta}+(B-1))^2}\right) + (B-2)\exp\left(\eta \frac{e^{\eta}+B-2}{(e^{\eta}+(B-1))^2}\right)}. \tag{26}$$

Now, taking the limit of (26) as η tends to infinity, we find that

$$\lim_{\eta\to\infty} \mathrm{SM}\left(\eta\big((1-\mathrm{SM}(\eta e_i)) \odot \mathrm{SM}(\eta e_j)\big)\right)_j \tag{27}$$

$$= \lim_{\eta\to\infty} \frac{\exp\left(\eta \frac{e^{\eta}(e^{\eta}+B-2)}{(e^{\eta}+(B-1))^2}\right)}{\exp\left(\eta \frac{B-1}{(e^{\eta}+(B-1))^2}\right) + \exp\left(\eta \frac{e^{\eta}(e^{\eta}+B-2)}{(e^{\eta}+(B-1))^2}\right) + (B-2)\exp\left(\eta \frac{e^{\eta}+B-2}{(e^{\eta}+(B-1))^2}\right)} = 1. \tag{28}$$

Since SM is continuous and maps into $\dot{\Delta}_B$ which is a closed set; then its limit must belong to the B-simplex also. However, if $\lim_{\eta\to\infty} \mathrm{SM}\left(\eta\big((1-\mathrm{SM}(\eta e_i)) \odot \mathrm{SM}(\eta e_j)\big)\right)_j$ and each coordinate of the limit is non-negative, we have that

$$\mathrm{SM}\left(\eta\big((1-\mathrm{SM}(\eta e_i)) \odot \mathrm{SM}(\eta e_j)\big)\right)_i$$

for every $i \in [B]_+ \setminus \{j\}$. Therefore, for all $\varepsilon > 0$, there exists some $\eta^{\star:3} > 0$ such that, if $\eta > \eta^{\star} \stackrel{\text{def}}{=} \max\{\eta^{\star:1}, \eta^{\star:2}, \eta^{\star:3}\}$ we have

$$\left\|\mathrm{SM}\left(\eta\big((1-\mathrm{SM}(\eta e_i)) \odot \mathrm{SM}(\eta e_j)\big)\right) - e_j\right\|_1 < \tfrac{\varepsilon}{2}. \tag{29}$$

Combining (25) and (29) yields (ii) and completes our proof.

Let $(\mathbf{E}_{i:1}, \mathbf{E}_{i:2}) \sim S_\eta(e_i, e_j)$. Define the disagreement set $A^\star$ as:

$$A^\star \stackrel{\text{def}}{=} \{\omega \in \Omega : (\mathbf{E}_{i:1}(\omega), \mathbf{E}_{i:2}(\omega)) \neq (e_i, e_j)\}.$$

Note that $A^\star$ is measurable since $\mathbf{E}_{i:j}$ are for $j \in \{1,2\}$ and since $\mathbb{R}^2 \ni (x,y) \mapsto I_{x=y} \in \{0,1\}$ is Borel. Now, for the last statement, we have by the dual representation of the TV-distance and by (ii)

$$\begin{aligned} \left\|\mathrm{Law}\left((\mathbf{E}_{i:1}, \mathbf{E}_{i:2})\right)(A^\star) - \delta_{(e_i,e_j)}(A^\star)\right\| &\leq \sup_A \left\|\mathrm{Law}\left((\mathbf{E}_{i:1}, \mathbf{E}_{i:2})\right)(A) - \delta_{(e_i,e_j)}(A)\right\| \\ &= \left\|\mathrm{Law}\left((\mathbf{E}_{i:1}, \mathbf{E}_{i:2})\right) - \delta_{(e_i,e_j)}\right\|_{TV} \\ &= \left\|(\mathbf{E}_{i:1}, \mathbf{E}_{i:2}) - (e_i, e_j)\right\|_1 \leq \varepsilon \end{aligned} \tag{30}$$

where the supremum is taken over all measurable sets A. Consequently, taking complements across (30) yields the final claim. □

Proof of Theorem 1. We recall that the recursion in (8) is indexed by $l = 1, \ldots, \Delta - 1$, so that for each $l \in \{1, \ldots, \Delta - 1\}$ the layer representations satisfy $\boldsymbol{x}^{(l)} \in \{0,1\}^{B_l}$ and $\boldsymbol{x}^{(l+1)} \in \{0,1\}^{B_{l+1}}$.

Accordingly, for each $l \in \{1, \dots, \Delta - 1\}$ and $j \in \{1, 2\}$, the random adjacency matrix $\boldsymbol{E}^{(l:j:e)}$ is a $B_{l+1} \times B_l$ matrix whose rows are one-hot almost surely; equivalently,

$$\boldsymbol{E}^{(l:j:e)}_{i:} \in \{e_1, \dots, e_{B_l}\} \quad \forall i \in [B_{l+1}]_+, \ \mathbb{P}\text{-a.s.}$$

In contrast, the transition from the raw input layer $l = 0$ to the lifted layer $l = 1$ is produced by the stochastic bit-lifting channel $\mathbf{X}^{\uparrow:W}$ (rather than by the matrices $\boldsymbol{E}^{(l:j:e)}$).

Fix $\Delta \in \mathbb{N}_+$, $\Upsilon \in \mathbb{N}_+$, and widths $B_{\text{in}} = B_0, B_1, \dots, B_\Delta = B_{\text{out}} = 1$ with $B_l \le \Upsilon$ for $l \in [\Delta]$. Let $\hat{\boldsymbol{f}}$ be defined by (8). We show that, $\mathbb{P}$-a.s. in ω, there exists $\mathcal{C}_\omega \in \text{CIRCUIT}^{B_{\text{in}}}_{\Delta,\Upsilon}$ such that $\hat{\boldsymbol{f}}(\omega, x) = f_{\mathcal{C}_\omega}(x)$ for all $x \in \{0,1\}^{B_{\text{in}}}$.

Step 1: A full-probability event on which all modules are Boolean. All random objects in the model are defined on a fixed probability space $(\Omega, \mathcal{F}, \mathbb{P})$. In particular: the lifted-input map takes values in the finite set $\{0,1\}^{B_1}$. Also, for each $l \in \{1, \dots, \Delta - 1\}$ and $j \in \{1, 2\}$, each row of every adjacency matrix $\boldsymbol{E}^{(l:j:e)}$ takes values in the finite set $\{e_1, \dots, e_{B_l}\}$, and each gate restriction belongs to the finite set $\mathbb{G}^\star$. Hence all random variables involved in the definitions below are finite-valued, and any event defined by constraints on their values is $\mathcal{F}$-measurable.

Let $\Omega_{\text{lift}} \subseteq \Omega$ be the event from Proposition 4 such that for every $\omega \in \Omega_{\text{lift}}$ and every input $x \in \{0,1\}^{B_{\text{in}}}$,

$$\mathbf{X}^{\uparrow:W}(\omega)(x) \in \{0,1\}^{B_1}.$$

Since the lifted-input map is finite-valued, $\Omega_{\text{lift}} \in \mathcal{F}$, and Proposition 4 implies that $\mathbb{P}(\Omega_{\text{lift}}) = 1$.

Let $\Omega_{\text{gate}} \subseteq \Omega$ be the event from Proposition 6 such that for every $\omega \in \Omega_{\text{gate}}$, every $l \in \{1, \dots, \Delta - 1\}$, and every node index $j \in \{1, \dots, B_{l+1}\}$, the restriction of the sampled activation map

$$\boldsymbol{\sigma}^{W^{(l:\sigma)}_j}(\omega)\big|_{\{0,1\}^2}$$

belongs to $\mathbb{G}^\star$, i.e., defines a meaningful Boolean gate on $\{0,1\}^2$. Since this restriction takes values in the finite set $\mathbb{G}^\star$, the event Ω_{gate} is $\mathcal{F}$-measurable. By Proposition 6, we have $\mathbb{P}(\Omega_{\text{gate}}) = 1$.

Finally, for each $l \in \{1, \dots, \Delta - 1\}$, the random adjacency matrices $\boldsymbol{E}^{(l:1:e)}$ and $\boldsymbol{E}^{(l:2:e)}$ select exactly one predecessor in layer l for every node in layer $l + 1$, i.e., each row is one-hot. Let $\Omega_{\text{edge}} \subseteq \Omega$ be the event on which, for every $l \in \{1, \dots, \Delta - 1\}$, $j \in \{1, 2\}$, and every $i \in [B_{l+1}]_+$,

$$\boldsymbol{E}^{(l:j:e)}_{i:}(\omega) \in \{e_1, \dots, e_{B_l}\}.$$

Since each adjacency row takes values in a finite set, $\Omega_{\text{edge}} \in \mathcal{F}$. By the assumptions on the adjacency matrices, $\mathbb{P}(\Omega_{\text{edge}}) = 1$.

Define

$$\Omega_{\text{all}} \stackrel{\text{def}}{=} \Omega_{\text{lift}} \cap \Omega_{\text{gate}} \cap \Omega_{\text{edge}}.$$

Then $\Omega_{\text{all}} \in \mathcal{F}$ and $\mathbb{P}(\Omega_{\text{all}}) = 1$. Fix any $\omega \in \Omega_{\text{all}}$ for the remainder of the proof. We construct a circuit $\mathcal{C}_\omega \in \text{CIRCUIT}^{B_{\text{in}}}_{\Delta,\Upsilon}$ computing $x \mapsto \hat{\boldsymbol{f}}(\omega, x)$.

Step 2: Construct the realized circuit $\mathcal{C}_\omega$. We build a layered DAG with layers

$$V_0 = \{v_{0,1}, \dots, v_{0,B_0}\}, \ V_1 = \{v_{1,1}, \dots, v_{1,B_1}\}, \ \dots, \ V_\Delta = \{v_{\Delta,1}\}, \tag{31}$$

so that layer V_0 corresponds to the input $x \in \{0,1\}^{B_0}$, and for $l \in \{1, \dots, \Delta\}$ layer V_l corresponds to the model state $\boldsymbol{x}^{(l)} \in \{0,1\}^{B_l}$.

Let $\text{val}_{\mathcal{C}_\omega}(v_{l,i}; x)$, for $l \in [\Delta]$ and $i \in [B_l]_+$, denote the output value of the i-th node in layer l of the layered DAG, evaluated at the input x, in the induced Boolean function $f_{\mathcal{C}_\omega}$ (corresponding to the node $v_{l,i}$ as in (31)). In particular, $\text{val}_{\mathcal{C}_\omega}(v_{\Delta,1}; x)$ corresponds to the output value of the induced Boolean function $f_{\mathcal{C}_\omega}(x) = f_\Delta(x)$ as in (1). We next define these values starting from the input layer.

Input values. For $x \in \{0,1\}^{B_{\text{in}}}$, define the value at the raw input nodes by

$$\text{val}_{\mathcal{C}_\omega}(v_{0,i}; x) \stackrel{\text{def}}{=} x_i \qquad \forall i \in \{1, \dots, B_0\}. \tag{32}$$

Lifted-layer values. For $x \in \{0,1\}^{B_{\text{in}}}$, define the value at $v_{1,j}$ to be

$$\text{val}_{\mathcal{C}_\omega}(v_{1,j}; x) \stackrel{\text{def}}{=} \boldsymbol{x}_j^{(1)}(\omega, x) = \mathbf{X}^{\uparrow:W}(\omega)(x)_j \in \{0,1\} \qquad \forall j \in \{1, \cdots, B_1\}, \tag{33}$$

using $\omega \in \Omega_{\text{lift}}$.

Lifting edges (layer $0 \to 1$). Fix $\omega \in \Omega_{\text{lift}}$. By definition of the lifting channel $\mathbf{X}^{\uparrow:W}$, there exists a (random) binary matrix $W(\omega) \in \{0,1\}^{B_1 \times 2B_0}$ such that, for every input $x \in \{0,1\}^{B_0}$, the lifted representation $\boldsymbol{x}^{(1)}(\omega, x)$ is obtained by selecting coordinates of the extended input vector $\begin{pmatrix} x \\ \mathbf{1}_{B_0} - x \end{pmatrix} \in \{0,1\}^{2B_0}$. In particular, for each $j \in \{1, \ldots, B_1\}$, the j-th row of $W(\omega)$ selects a unique index $q(j;\omega) \in \{1, \ldots, 2B_0\}$.

Accordingly, define for each $j \in \{1, \ldots, B_1\}$ the associated raw index $i(j;\omega) \in \{1, \ldots, B_0\}$ by

$$i(j;\omega) \stackrel{\text{def}}{=} \begin{cases} q(j;\omega), & q(j;\omega) \le B_0, \\ q(j;\omega) - B_0, & q(j;\omega) > B_0. \end{cases}$$

We then record the underlying raw coordinate by adding the directed edge

$$\big(v_{0,i(j;\omega)},\, v_{1,j}\big).$$

Here the edge identifies the raw index $i(j;\omega)$.

Note that the values at layer V_1 are still defined by (33); the edges from V_0 to V_1 encode the correspondence between lifted bits and raw input coordinates induced by $W(\omega)$.

Edges. For each $l \in \{1, \ldots, \Delta - 1\}$ and each node index $j \in \{1, \ldots, B_{l+1}\}$, since $\omega \in \Omega_{\text{edge}}$, the j-th row of $\boldsymbol{E}^{(l:1:e)}(\omega)$ is one-hot, so it selects a unique index $p_1(l, j; \omega) \in \{1, \ldots, B_l\}$; similarly the j-th row of $\boldsymbol{E}^{(l:2:e)}(\omega)$ selects a unique $p_2(l, j; \omega) \in \{1, \ldots, B_l\}$, Add directed edges

$$\big(v_{l,p_1(l,j;\omega)},\, v_{l+1,j}\big), \qquad \big(v_{l,p_2(l,j;\omega)},\, v_{l+1,j}\big).$$

Thus, every node in layers V_l for $l \ge 2$ has fan-in exactly 2, edges occur only between consecutive layers, and the edges from V_0 to V_1 encode the correspondence induced by the lifting map.

Gates. For each $l \in \{1, \ldots, \Delta - 1\}$ and each $j \in \{1, \ldots, B_{l+1}\}$, assign to node $v_{l+1,j}$ the Boolean gate

$$g_{l+1,j} \stackrel{\text{def}}{=} \boldsymbol{\sigma}^{W_j^{(l:\sigma)}}(\omega)\big|_{\{0,1\}^2} \in \mathbb{G}^\star,$$

using $\omega \in \Omega_{\text{gate}}$.

Definition of the realized circuit. The realized Boolean circuit associated with $\omega \in \Omega_{\text{all}}$ is the triple $\mathcal{C}_\omega \stackrel{\text{def}}{=} (V, E, g.)$, where:

- The vertex set is $V \stackrel{\text{def}}{=} \bigsqcup_{l=0}^{\Delta} V_l$ with V_l defined as in (31).
- The edge set is

$$E \stackrel{\text{def}}{=} \Big(\bigcup_{j=1}^{B_1} \{(v_{0,i(j;\omega)},\, v_{1,j})\} \Big) \cup \Big(\bigcup_{l=1}^{\Delta-1} \bigcup_{j=1}^{B_{l+1}} \big\{(v_{l,p_1(l,j;\omega)},\, v_{l+1,j}), (v_{l,p_2(l,j;\omega)},\, v_{l+1,j})\big\} \Big).$$

- The gate assignment is $g. \stackrel{\text{def}}{=} \{\, g_{l+1,j} : \{0,1\}^2 \to \{0,1\} \,\}_{l=1,\ldots,\Delta-1;\, j=1,\ldots,B_{l+1}}$.

This defines a Boolean circuit $\mathcal{C}_\omega = (V, E, g.)$ of depth Δ and width at most $\max_{0 \le l \le \Delta} B_l \le \Upsilon$.

Step 3: Prove that $\mathcal{C}_\omega$ computes $\hat{\boldsymbol{f}}(\omega, \cdot)$ (induction on depth). We prove that for every $l \in \{1, \ldots, \Delta\}$, every input $x \in \{0,1\}^{B_{\text{in}}}$, and every node index $j \in \{1, \ldots, B_l\}$,

$$\text{val}_{\mathcal{C}_\omega}(v_{l,j}; x) = \boldsymbol{x}_j^{(l)}(\omega, x). \tag{34}$$

Base case ($l = 1$). By definition of the input layer and the lifted-input channel,

$$\mathrm{val}_{\mathcal{C}_\omega}(v_{1,j}; x) \stackrel{\mathrm{def}}{=} \boldsymbol{x}_j^{(1)}(\omega, x) = \mathbf{X}^{\uparrow:W}(\omega)(x)_j \in \{0,1\} \qquad \forall j \in \{1, \cdots, B_1\}$$

Hence, (34) holds for $l = 1$ (see (33)).

Induction hypothesis. Assume that for some $l \in \{1, \ldots, \Delta - 1\}$, (34) holds for all nodes in layer l, i.e.

$$\mathrm{val}_{\mathcal{C}_\omega}(v_{l,j}; x) = \boldsymbol{x}_j^{(l)}(\omega, x) \quad \forall j \in \{1, \ldots, B_l\},\ \forall x \in \{0,1\}^{B_{\mathrm{in}}}.$$

Inductive step. Fix an arbitrary node index $j \in \{1, \ldots, B_{l+1}\}$ and input $x \in \{0,1\}^{B_{\mathrm{in}}}$. By construction of $\mathcal{C}_\omega$,

$$\mathrm{val}_{\mathcal{C}_\omega}(v_{l+1,j}; x) = g_{l+1,j}\Big(\mathrm{val}_{\mathcal{C}_\omega}(v_{l,p_1(l,j;\omega)}; x), \mathrm{val}_{\mathcal{C}_\omega}(v_{l,p_2(l,j;\omega)}; x)\Big).$$

By the induction hypothesis,

$$\mathrm{val}_{\mathcal{C}_\omega}(v_{l,p_i(l,j;\omega)}; x) = \boldsymbol{x}^{(l)}_{p_i(l,j;\omega)}(\omega, x), \qquad i = 1, 2.$$

Since $g_{l+1,j} = \boldsymbol{\sigma}^{W_j^{(l:\sigma)}}(\omega)\big|_{\{0,1\}^2}$ and the one-hot rows of $\boldsymbol{E}^{(l:1:e)}(\omega)$ and $\boldsymbol{E}^{(l:2:e)}(\omega)$ select exactly the predecessors $p_1(l, j; \omega)$ and $p_2(l, j; \omega)$, the recursion (8) implies

$$\boldsymbol{x}_j^{(l+1)}(\omega, x) = \boldsymbol{\sigma}^{W_j^{(l:\sigma)}}(\omega)\Big(\boldsymbol{x}^{(l)}_{p_1(l,j;\omega)}(\omega, x), \boldsymbol{x}^{(l)}_{p_2(l,j;\omega)}(\omega, x)\Big).$$

Thus

$$\mathrm{val}_{\mathcal{C}_\omega}(v_{l+1,j}; x) = \boldsymbol{x}_j^{(l+1)}(\omega, x).$$

Since j was arbitrary, (34) holds for all nodes in layer $l + 1$. This completes the induction.

Step 4: Final conclusion. Taking $l = \Delta$ in (34) and recalling $B_\Delta = 1$, we have for every input $x \in \{0,1\}^{B_{\mathrm{in}}}$,

$$\hat{\boldsymbol{f}}(\omega, x) = \boldsymbol{x}^{(\Delta)}(\omega, x) = \mathrm{val}_{\mathcal{C}_\omega}(v_{\Delta,1}; x),$$

so $\hat{\boldsymbol{f}}(\omega, \cdot)$ is computed by the Boolean circuit $\mathcal{C}_\omega$ of depth Δ and width $\leq \Upsilon$. Therefore $\hat{\boldsymbol{f}}(\omega, \cdot) \in \mathrm{CIRCUIT}^{B_{\mathrm{in}}}_{\Delta,\Upsilon}$ for every $\omega \in \Omega_{\mathrm{all}}$. Since $\mathbb{P}(\Omega_{\mathrm{all}}) = 1$, Equivalently, for every parameter setting θ, the induced law μ_θ is supported on $\mathrm{CIRCUIT}^{B_{\mathrm{in}}}_{\Delta,\Upsilon}$:

$$\mu_\theta\Big(\mathrm{CIRCUIT}^{B_{\mathrm{in}}}_{\Delta,\Upsilon}\Big) = 1.$$

This completes the proof. □

E.1 Tree Construction

We begin with the worst-case representation of a Boolean function as a tree whose leaves are literals in the Boolean cube, their negations, and possibly repeated copies of each. Moreover, the circuit is implemented as a full binary tree, so it has the exact desired multipartite structure: every internal node is a binary, single-output gate of type AND, OR, or PROJ_A. The term *single-output* describes the scalar value computed by a gate and does not prohibit downstream reuse. This avoids the complications that arise with circuit representations whose computational graph differs from that of our neural networks, or which mix unary and binary gates at internal nodes and therefore yield an incompatible computational structure.

Lemma 4 (Complete Binary Tree Representation of Boolean Functions)**.** *Let $B \in \mathbb{N}_+$ and let $f : \{0,1\}^B \to \{0,1\}$ be a Boolean function. There exist integers $B^\uparrow, \Delta \in \mathbb{N}_+$ and a Boolean circuit $\mathcal{C}$ computing f such that:*

(i) the underlying DAG of $\mathcal{C}$ is a rooted complete binary tree of depth Δ,

(ii) the $B^{\uparrow}$ leaves are labelled by literals in $\{x_1, \ldots, x_B, \neg x_1, \ldots, \neg x_B\}$, with repetitions allowed; i.e. for every $\tilde{b} \in [B^{\uparrow}]_+$ we have

$$x_{\tilde{b}} \in \{x_b\}_{b=1}^{B} \cup \{\neg x_b\}_{b=1}^{B},$$

(iii) every gate in $\mathcal{C}$ belongs to $\{\mathrm{OR}, \mathrm{AND}, \mathrm{PROJ}_A\} = \{g_8, g_2, g_4\}$.

Moreover, the number of gates is exactly $B^{\uparrow} - 1 \leq 2B2^B - 1$ and

$$B^{\uparrow} \leq 2B2^B, \qquad \Delta \leq \lceil \log_2(B2^B) \rceil \leq B + \lceil \log_2 B \rceil. \tag{35}$$

Proof location. See page 27.

We now sharpen the previous result to functions which are *possibly* "sparse" in a Boolean sense.

Definition 4 (S-Junta)**.** A Boolean function $f : \{0,1\}^B \to \{0,1\}$ is said to *depend on the ith variable* if there exist inputs $x, y \in \{0,1\}^B$ which differ only in their ith coordinate and satisfy $f(x) \neq f(y)$. Equivalently, we say that the ith variable is *relevant* to f. If f has at most s relevant variables, then we call f an *s-junta*.

Clearly a B-Junta on B bits is just an arbitrary Boolean function on $\{0,1\}^B$. However, at the other extreme of the spectrum we have dictator functions; defined as follows.

Example 7 (Dictator Functions)**.** Let $B \in \mathbb{N}_+$. A Boolean function $f : \{0,1\}^B \to \{0,1\}$ is called a dictator if there exists some $b \in [B]_+$ such that: either $f(x) = x_b$ or $f(x) = \neg x_b$ for every $x \in \{0,1\}^B$. Dictator functions are precisely 1-juntas.

The following equivalent formulation of a junta will be more handy for us.

Lemma 5 (S-Juntas as Embedded Permutations)**.** *Let $B, S \in \mathbb{N}_+$ with $S \leq B$. A Boolean function $f : \{0,1\}^B \to \{0,1\}$ is an S-junta iff there exist distinct $(i_1, \ldots, i_S) \in [B]_+$ and some $f_S : \{0,1\}^S \to \{0,1\}$ such that*

$$f(x) = f_S\big(x_{i_1}, \ldots, x_{i_S}\big) \quad \text{for all } x \in \{0,1\}^B.$$

Equivalently, letting $\Pi \in \{0,1\}^{S \times B}$ be the selection matrix with $(\Pi x)_j = x_{i_j}$ for each $j \in [S]_+$ and for every $x \in \{0,1\}^B$ we have

$$f(x) = f_S(\Pi x). \tag{36}$$

Proof location. See page 28.

A direct consequence of Lemmata 4 and 5 implies the following sharper result.

Lemma 6. (Construction of Low-Dimensional Complete Boolean Circuit on Lifted Literals) *Let $S, B \in \mathbb{N}_+$ with $S \leq B$, and let $f : \{0,1\}^B \to \{0,1\}$ be an S-Junta. Then, there exists a $B^{\uparrow} \in \mathbb{N}_+$, a stochastic matrix $\Pi \in \{0,1\}^{B^{\uparrow} \times B}$, a Boolean function $f_S : \{0,1\}^{B^{\uparrow}} \to \{0,1\}$, integers $B^{\uparrow}, \Delta \in \mathbb{N}_+$, and a Boolean circuit $\mathcal{C}$ such that*

$$f = f_S \circ (\Pi \cdot) \tag{37}$$

for every $x \in \{0,1\}^B$ and the following hold:

(i) $\mathcal{C}$ computes f_S,

(ii) the underlying DAG of $\mathcal{C}$ is a rooted complete binary tree of depth Δ,

(iii) the $B^{\uparrow}$ leaves are labelled by literals in $\{x_1, \ldots, x_B, \neg x_1, \ldots, \neg x_B\}$, with repetitions allowed; i.e. for every $\tilde{b} \in [B^{\uparrow}]_+$ we have

$$x_{\tilde{b}} \in \{x_b\}_{b=1}^{B} \cup \{\neg x_b\}_{b=1}^{B},$$

(iv) every gate in $\mathcal{C}$ belongs to $\{\mathrm{OR}, \mathrm{AND}, \mathrm{PROJ}_A\} = \{g_8, g_2, g_4\}$.

The number of gates is exactly $B^{\uparrow} - 1 \leq 2S2^S - 1$ and $B^{\uparrow} \leq 2S2^S$, and $\Delta \leq S + \lceil \log_2 S \rceil$. If, moreover, there exists some $C \in \mathbb{N}$ such that $S \leq C\lceil \log(B) \rceil$ such that

$$S \leq C\lceil \log(B) \rceil \tag{38}$$

then there exists some $p, c \in \mathbb{N}_+$ independent of B such that $B^{\uparrow} \leq c\, B^p \lceil \log_2(B) \rceil$.

Proof location. See page 28.

E.1.1 Proofs of Results from Subsection E.1

Proof of Lemma 4. If $f = 0$ is the constant false function, then we may take the formula $x_1 \text{ AND } \neg x_1$ and skip directly to the padding step below, with $N_{\text{lit}} = 2 \leq B2^B$; cf. (40). Whence, we henceforth assume that $f \neq 0$.

Step 1: DNF with a bounded number of literal occurrences. Write f in canonical disjunctive normal form (CDN). For each $a = (a_1, \ldots, a_B) \in \{0,1\}^B$ with $f(a) = 1$, define $T_a : \{0,1\}^B \to \{0,1\}$ for every $x \in \{0,1\}^B$ by

$$T_a(x) \stackrel{\text{def}}{=} \bigwedge_{i=1}^{B} \ell_i^{(a)}(x), \text{ where } \ell_i^{(a)}(x) \stackrel{\text{def}}{=} \begin{cases} x_i, & a_i = 1, \\ \neg x_i, & a_i = 0 \end{cases} \tag{39}$$

for every $i \in [B]_+$. Then, for every $x \in \{0,1\}^B$, $f(x) = \bigvee_{\{a: f(a)=1\}} T_a(x)$. Let T be the number of satisfying assignments of f. Then $1 \leq T \leq 2^B$, and each T_a uses exactly B literals, so the total number of literal occurrences N_{lit} satisfies

$$N_{\text{lit}} \stackrel{\text{def}}{=} TB \leq B2^B. \tag{40}$$

Thus we obtain a formula over $\{\text{AND}, \text{OR}\}$ with at most N_{lit} literal leaves.

Step 2: Conversion into a Binary Tree.
Remark: *We now replace the formula from Step* 1 *by an equivalent binary formula whose depth we can bound explicitly.*

Fix a with $f(a) = 1$. We claim that the conjunction $T_a(x) = \bigwedge_{i=1}^{B} \ell_i^{(a)}(x)$ can be computed by a circuit whose underlying DAG is a binary tree of AND-gates of depth at most $\lceil \log_2 B \rceil$, with leaves given by the literals $\ell_1^{(a)}(x), \ldots, \ell_B^{(a)}(x)$. To see this, we define the circuit for T_a level-by-level: For the 0th level, the nodes are the B input literals $z_1^{(0)}, \ldots, z_B^{(0)}$, where for every $i \in [B]_+$, we have set $z_i^{(0)} \stackrel{\text{def}}{=} \ell_i^{(a)}(x)$.

We proceed iteratively for level $k \in \mathbb{N}_+$ large enough. Suppose that level $k-1$ has m_{k-1} nodes and denote $z_1^{(k-1)}, \ldots, z_{m_{k-1}}^{(k-1)}$. Now, for each $j \in [\lfloor m_{k-1}/2 \rfloor]_+$, we introduce an AND-gate with inputs $z_{2j-1}^{(k-1)}$ and $z_{2j}^{(k-1)}$, and denote its output by $z_j^{(k)} \stackrel{\text{def}}{=} z_{2j-1}^{(k-1)} \wedge z_{2j}^{(k-1)}$. If m_{k-1} is even, we are done, otherwise there is exactly one remaining node $z_{m_{k-1}}^{(k-1)}$ which has not been paired. In this case, set $z_{m_k}^{(k)} \stackrel{\text{def}}{=} z_{m_{k-1}}^{(k-1)}$; where $m_k \stackrel{\text{def}}{=} \lfloor m_{k-1}/2 \rfloor + 1$. Thus,

$$m_k \leq \left\lceil \frac{m_{k-1}}{2} \right\rceil. \tag{41}$$

Since $m_0 = B$, our recursion yielded

$$m_k \leq \left\lceil \frac{B}{2^k} \right\rceil. \tag{42}$$

Therefore, for $K \stackrel{\text{def}}{=} \lceil \log_2 B \rceil$ we have $m_K \leq \left\lceil \frac{B}{2^{\lceil \log_2 B \rceil}} \right\rceil = 1$. In particular, after at most K levels, there is exactly one remaining node $z_1^{(K)}$.

By construction, every $z_j^{(k)}$ is the conjunction of a subset of the literals $\{\ell_i^{(a)}(x)\}_{i=1}^{B}$, and these subsets form a partition of $\{1, \ldots, B\}$ across all nodes at level k. Hence the final node $z_1^{(K)}$ is equal to

$$z_1^{(K)} = \bigwedge_{i=1}^{B} \ell_i^{(a)}(x) = T_a(x). \tag{43}$$

Moreover, every internal node in this construction has fan-in 2, so the underlying DAG is a binary tree whose depth is at most $K = \lceil \log_2 B \rceil$.

Alternatively, if we have a disjunction of the form

$$f(x) = \bigvee_{\{a:\, f(a)=1\}} T_a(x).$$

There are T such terms. Applying exactly the same construction as above but replacing AND by OR and the inputs $z_j^{(0)}$ by the $T_a(x)$, we obtain a binary tree of OR-gates which computes $\bigvee_{\{a:\, f(a)=1\}} T_a(x)$ and has depth at most $\lceil \log_2 T \rceil$.

Putting it all together; upon composing the AND-trees for each T_a with the top-level OR-tree, we obtain a binary formula tree T_0 computing f such that:

(a) the leaves of T_0 are exactly the literal occurrences $\ell_i^{(a)}(x)$ from the DNF in Step 1, so their total number is

$$N_{\text{lit}} \leq B\, 2^B, \tag{44}$$

(b) Since $T \leq 2^B$ then the depth of T_0 is at most

$$\lceil \log_2 B \rceil + \lceil \log_2 T \rceil \leq \lceil \log_2 B \rceil + B. \tag{45}$$

This T_0 is the binary tree we will pad in the next step.

Step 3: Conversion to a complete binary tree via projection gates

$$\Delta \stackrel{\text{def}}{=} B + \lceil \log_2 B \rceil \text{ and } B^{\uparrow} \stackrel{\text{def}}{=} 2^{\Delta}. \tag{46}$$

Thus, the depth of T_0, denoted by $\text{depth}(T_0)$ satisfies $\text{depth}(T_0) \leq \Delta$ and every leaf of T_0 lies at some depth $\ell \leq \Delta$. For any leaf v of T_0 lying a level/layer of depth $\ell < \Delta$ with $\ell \in \mathbb{N}$. We iteratively replace v by a new internal gate u decorated as PROJ_A, then, we may make v the left child of u, and add a new leaf w as the right child of u, labelled by any literal in $\{x_1, \ldots, x_B, \neg x_1, \ldots, \neg x_B\}$. Then the value at u equals the value at v, since $\text{PROJ}_A(x, y) = x$ ignores its second input. This increases the depth of that branch by 1 and increases the total number of leaves by 1.

Iterating this operation on each leaf with depth $< \Delta$ until all leaves lie at depth Δ yields a new tree T_1 such that: 1) every internal node has two children (the original AND, OR gates and the newly added PROJ_A gates), and 2) every leaf lies at depth exactly Δ. Hence T_1 is a complete binary tree of depth Δ. Any finite full, i.e. every internal node has out-degree 2, binary tree of depth Δ has exactly 2^{Δ} leaves. Consequently, $B^{\uparrow} = 2^{\Delta}$ and the root of T_1 still computes f, all leaves are labelled by literals as in (ii), and all gates belong to $\{\text{OR}, \text{AND}, \text{PROJ}_A\}$ as in (iii). Finally, by the choice of Δ,

$$B^{\uparrow} = 2^{\Delta} = 2^{B + \lceil \log_2 B \rceil} = 2^B\, 2^{\lceil \log_2 B \rceil} \leq 2^B (2B) = 2B2^B.$$

Now, since $\log_2(B2^B) = B + \log_2 B$, then we also find that $\Delta = B + \lceil \log_2 B \rceil \leq \lceil \log_2(B2^B) \rceil$.

Step 4: Final Counts. It remains to count the number of gates. Let I denote the number of internal nodes of T_1, i.e. those nodes decorated by gates. Counting edges in two ways gives

$$2I = \#\{\text{edges}\} = I + B^{\uparrow} - 1.$$

Consequently, $B^{\uparrow} = I + 1$. Whence, $I = B^{\uparrow} - 1 \leq 2B2^B - 1$, which is the number of gates of $\mathcal{C}$. □

Proof of Lemma 5. If f is an S-junta, there is a set $J \subseteq [B]_+$ of relevant variables with $|J| \leq S$. Enlarge J to $\{i_1, \ldots, i_S\}$ if needed. Define $f_S : \{0,1\}^S \to \{0,1\}$ by $f_S(z_1, \ldots, z_S) \stackrel{\text{def}}{=} f(x)$ for any x with $x_{i_j} = z_j$; this is well-defined because f does not depend on coordinates outside $\{i_1, \ldots, i_S\}$. Then $f(x) = f_S(x_{i_1}, \ldots, x_{i_S})$ for all x. Conversely, if $f(x) = f_S(x_{i_1}, \ldots, x_{i_S})$, then changing any coordinate outside $\{i_1, \ldots, i_S\}$ leaves $f(x)$ unchanged, so f has at most S relevant variables; whence, f is an S-junta. □

Proof of Lemma 6. All but the last claims are a direct consequence of Lemmata 4 and 5. For the final claim, note that if there exists some $C \in \mathbb{N}$ such that $S \leq C\lceil \log(B) \rceil$ then (35) implies that

$$B^{\uparrow} \leq S2^S \leq C\lceil \log_2 B \rceil\, 2^{C\lceil \log_2 B \rceil} \leq C\, 2^C\, B^C\, \lceil \log_2 B \rceil.$$

Setting $p \stackrel{\text{def}}{=} C$ and $c \stackrel{\text{def}}{=} C2^C$ yields the result. □

We first prove a general construction for S-juntas. Theorem 2 follows by taking $S = B$, since every Boolean function on B bits is a B-junta. Theorem 3 follows by applying the same construction under the additional assumption $S \leq C \log B$ and simplifying the resulting size bounds.

Proposition 9 (Universal construction for S-juntas)**.** *Let $B, S \in \mathbb{N}_+$ and let $f : \{0,1\}^B \to \{0,1\}$ be an S-junta. For every $0 < \delta \leq 1$, there exists an SBC $\hat{\boldsymbol{f}} : \Omega \times \{0,1\}^B \to \{0,1\}$ of the form* (8) *such that*

$$\mathbb{P}\Big(\hat{\boldsymbol{f}}(x) = f(x)\ \forall x \in \{0,1\}^B\Big) \geq 1 - \delta.$$

Moreover, the construction can be chosen with the depth, width, and bit-lifting size bounds stated in Lemma 6.

Proof of Proposition 9. Fix our failure probability $\delta \in (0, 1]$.

Step 0 **- Setup:**

Fix a temperature hyperparameter $\eta > 0$, to be determined retroactively. Since $f : \{0,1\}^B \to \{0,1\}$ is an S-junta, Lemma 6 gives a complete binary formula-tree representation of f as in (37).

Now the computation graph $G = (V, E)$ of the circuit $\mathcal{C}$ computing f_S, in the notation of Lemma 6, is a rooted complete binary tree of depth $\Delta \in \mathbb{N}_+$. Therefore, we may decompose V as

$$V = \bigsqcup_{l=0}^{\Delta} V_l \text{ where } V_l = \{v_{l:i}\}_{i=1}^{2^{\Delta-l}} \tag{47}$$

and we may partition E as

$$E = \bigsqcup_{l=1}^{\Delta} E_l \text{ where } E_l = \bigsqcup_{i=1}^{2^{\Delta-l}} \big\{(v_{l-1:i_1}, v_{l:i}), (v_{l-1:i_2}, v_{l:i})\big\}_{i=1}^{2^{\Delta-l}} \tag{48}$$

where, for every $i \in [2^{\Delta-l}]_+$ the indices $i_1, i_2 \in [2^{\Delta-l+1}]_+$ in (48) are distinct and unique. Moreover, WLOG, we may arrange $v_{l:i}$ the edges E in the graph G do not cross, as trees are planar.

Furthermore, since the gates in $\mathcal{C}$ only decorate the vertices in $\sqcup_{l=1}^{\Delta} V_l$ and not in V_0 (those are only variables described by Lemma 6 (iii)). We denote the gate decorating the vertex $v_{l:i}$ by $g_{l:i}$ and, WLOG, we identify $g_{l:i} \in \{g_k\}_{k=1}^{K}$ (say $g_{l:i} = g_k$) with the unique standard basis vector in $\{e_k\}_{k=1}^{16}$ with 1 only on the entry of the index k).

Step 1 **- Representation:**

Consider an $\hat{\boldsymbol{f}}$ with representation (8) having depths $(2^\Delta, 2^{\Delta-1}, \ldots, 2^0) = (2^{\Delta-1})_{l=0}^{\Delta} \in \mathbb{N}_+^{\Delta+1}$; we determine its internal parameters now. Now for every $l \in [\Delta]_+$ and each $i \in [2^{\Delta-l}]_+$ let

$$W^{(l:\sigma)} \stackrel{\text{def}}{=} \big(\eta\, g_{l:i}\big)_{i=1}^{2^{\Delta-l}},\ W^{(l:1)} \stackrel{\text{def}}{=} \eta \oplus_{i=1}^{2^{\Delta-l}} e_{i_1}, \text{and } W^{(l:2)} \stackrel{\text{def}}{=} \eta \oplus_{i=1}^{2^{\Delta-l}} e_{i_2}. \tag{49}$$

Now applying Propositions 7 and 5 and taking a union bound, the probability that $\hat{\boldsymbol{f}}(x, \omega) = f(x)$ for every element of the Boolean cube $x \in \{0,1\}^{B^\uparrow}$ is at least

$$1 - \sum_{l=1}^{\Delta} \sum_{i=1}^{2^{\Delta-l}} \tilde{\delta}(\eta) \tag{50}$$

where $\tilde{\delta}(\eta) \in (0, 1]$ depends continuously on $\eta > 0$; with $\lim_{\uparrow\infty} \tilde{\delta}(\eta) = 0$. Now, applying Proposition 4, we may choose $W \in \mathbb{R}^{B^\uparrow \times 2B}$, and thus, $X^{\uparrow:W}$ as defined in (3), so that $X^{\uparrow:W}$ coincides with Π , with probability at least

$$1 - \bar{\delta}(\eta) \tag{51}$$

for some $\bar{\delta}(\eta)$ depending continuously on η and such that $\lim_{\eta\uparrow\infty} \bar{\delta}(\eta) = 0$. Consequently, we may choose η large enough so that

$$\max\big\{\bar{\delta}(\eta), \tilde{\delta}(\eta)\big\} \leq \delta \left(\max\Big\{2, 2^\Delta - 1\Big\}\right)^{-1}$$

where we note that $\sum_{l=1}^{\Delta} \sum_{i=1}^{2^{\Delta-l}} 1 = 2^\Delta - 1$. Taking another union bound, our choice of η and the lower-bounds in (50) and (51) yield the conclusion. □

Proof of Theorem 2. Every Boolean function $f : \{0,1\}^B \to \{0,1\}$ is trivially a B-junta. Applying Proposition 9 with $S = B$ gives an SBC of the form (8) such that

$$\mathbb{P}\Big(\hat{\boldsymbol{f}}(x) = f(x)\ \forall x \in \{0,1\}^B\Big) \geq 1 - \delta.$$

This is exactly (9). □

Proof of Theorem 3. Let $f : \{0,1\}^B \to \{0,1\}$ be an S-junta with $S \leq C \log B$. By Proposition 9, there exists an SBC satisfying

$$\mathbb{P}\Big(\hat{\boldsymbol{f}}(x) = f(x)\ \forall x \in \{0,1\}^B\Big) \geq 1 - \delta.$$

The same proposition gives the depth, width, neuron-count, and bit-lifting bounds in terms of S. Since $S \leq C \log B$, Lemma 6 implies that $B^{\uparrow} \leq S2^S = O(B^C \log B)$. Therefore, $\Delta \in O(\log B + \log\log B)$ and $B^{\uparrow} \in O(B^C \log B)$. In particular, if $0 < C \leq 1$, then $N, B^{\uparrow} \in O(B \log B)$.

Since the Boolean circuit $\mathcal{C}$ computing f, given by Lemma 6, is a tree then its width is precisely equal to the number of its leaves; whence $\Upsilon = B^{\uparrow}$. Moreover, since the underlying graph of $\mathcal{C}$ is a complete binary tree then the number of nodes (which coincides with the number of neurons) is $N = \sum_{j=1}^{\Delta} 2^j = 2^{\Delta+1} - 1 = 2\Upsilon - 1 \in \mathcal{O}(\Upsilon) = \mathcal{O}(B^C \log(B))$. □

E.2 Proof of Corollary 1

Proof. Fix $x \in \{0,1\}^{B_{\text{in}}}$. The event that the sampled circuit is wrong at this particular input is contained in the event that it fails to compute f on the whole Boolean cube:

$$\{\, f_{\mathcal{C}}(x) \neq f(x) \,\} \subseteq E_f^c.$$

Since $\mu_\theta(E_f) \geq 1 - \delta$, this gives

$$\mathbb{P}_{\mathcal{C}\sim\mu_\theta}\big(f_{\mathcal{C}}(x) \neq f(x)\big) \leq \delta.$$

If $f(x) = 1$, then

$$1 - p_\theta(x) = \mathbb{P}_{\mathcal{C}\sim\mu_\theta}\big(f_{\mathcal{C}}(x) = 0\big) \leq \delta,$$

so $|p_\theta(x) - f(x)| \leq \delta$. If $f(x) = 0$, then

$$p_\theta(x) = \mathbb{P}_{\mathcal{C}\sim\mu_\theta}\big(f_{\mathcal{C}}(x) = 1\big) \leq \delta,$$

so again $|p_\theta(x) - f(x)| \leq \delta$. Since x was arbitrary,

$$\|p_\theta - f\|_{\ell_\infty(\{0,1\}^{B_{\text{in}}})} \leq \delta.$$

If $\delta < 1/2$, then $p_\theta(x) > 1/2$ whenever $f(x) = 1$ and $p_\theta(x) < 1/2$ whenever $f(x) = 0$, so thresholding at $1/2$ recovers the full truth table. □

F Experimental Setup and Implementation Details

This appendix documents the experimental protocol and implementation-facing details used to obtain the numerical results reported in the main text. The theoretical results in Sections 3–4 are agnostic to these choices; the goals here are (i) reproducibility and (ii) a precise description of the evaluation and interpretability probes used for continuous baselines.

F.1 Task family and dataset generation

Each instance is a Boolean function $f : \{0,1\}^B \to \{0,1\}$ specified by a syntactic formula generated by the benchmark expression sampler. The main benchmark contains 2000 tasks: for each $B \in \{2, 4, 6, \ldots, 20\}$, we sample 200 formulas. For every task we train and evaluate on the full truth table, i.e., all 2^B assignments, and report exact match (EM), meaning that the predicted truth table equals the target truth table on all rows.

The sampled formulas are stored in their original unsimplified syntax. Logical simplification is used only to record metadata and is never used to modify the training labels. In particular, the ambient input dimension B may exceed the number of semantically active variables after reduction, so the benchmark tests realization on the full ambient truth table.

Structural proxies. Each row also provides generation-time proxies S_{base} and L_{base}, used only to set model budgets. S_{base} is a width proxy from formula generation and L_{base} is a heuristic depth proxy. These proxies are informative but not privileged: models are trained only from truth-table input-output labels.

F.2 Proof-Level Marginals and Differentiable-Relaxation Metrics

For a target Boolean function $f : \{0,1\}^B \to \{0,1\}$ and an SBC parameter setting θ, the proof-level model induces a random-circuit law μ_θ and the exact predictive marginal

$$p_\theta(x) = \mathbb{P}_{\mathcal{C}\sim\mu_\theta}\big(f_{\mathcal{C}}(x) = 1\big).$$

We reserve *random-circuit marginal* for this expectation over discrete circuits. The experimental implementation does not enumerate or sample all circuits to evaluate p_θ. Instead, it deterministically propagates soft routing weights and soft gate weights through differentiable gate interpolants. We denote the resulting relaxed score by $\hat{p}_\theta(x) \in [0,1]$. It is a differentiable relaxation used for optimization and is not, in general, equal to $p_\theta(x)$.

Truth-table exact match. The primary metric in the main text is exact match after thresholding the relaxed score:

$$\hat{f}_\theta(x) = \mathbf{1}\{\hat{p}_\theta(x) \geq 1/2\}, \qquad \mathrm{EM}(\theta; f) = \mathbf{1}\Big[\hat{f}_\theta(x) = f(x)\ \forall x \in \{0,1\}^B\Big].$$

For SBC, this is relaxed exact match. For MLPs, the same formula is applied to the thresholded real-valued predictor.

Relaxed disagreement score. Beyond thresholded exact match, the relaxed row-wise error is

$$\mathcal{R}_{\text{relax}}(\theta; f) \stackrel{\text{def}}{=} 2^{-B} \sum_{x\in\{0,1\}^B} \Big[f(x)\big(1-\hat{p}_\theta(x)\big)+\big(1-f(x)\big)\hat{p}_\theta(x)\Big] = 2^{-B} \sum_{x\in\{0,1\}^B} |\hat{p}_\theta(x)-f(x)|.$$

This quantity evaluates the deterministic differentiable relaxation. Only in the special case $\hat{p}_\theta = p_\theta$ does it equal the expected truth-table error of a circuit sampled from μ_θ:

$$\mathbb{E}_{\mathcal{C}\sim\mu_\theta}\left[2^{-B} \sum_{x\in\{0,1\}^B} \mathbf{1}\{f_{\mathcal{C}}(x) \neq f(x)\}\right].$$

F.3 Models and Matching Regimes

SBC. We use the differentiable 16-gate SBC model throughout; every discrete gate in its dictionary is binary and single-output. For each task, we instantiate one canonical reference SBC predictor with

$$S_{\text{model}} = S_{\text{base}} + 10, \qquad L_{\text{model}} = L_{\text{base}}.$$

The implementation uses softmax-based gate/routing distributions, learned routing, and no bit-lifting in the main comparison. This single task-wise SBC reference is the object against which all MLP baselines are matched. MAP-decoding diagnostics are reported separately in Appendix G.3.

MLP baselines. We train ReLU MLPs on the same full truth tables. For each task, all MLP variants are matched against the same task-wise SBC reference. We consider two matching rules:

- **Neuron-match:** the MLP hidden width and depth are set to the SBC width and depth.
- **Param-match:** the MLP depth is fixed to the SBC depth, and its hidden width is chosen so that the MLP trainable parameter count does not exceed the trainable parameter count of the corresponding SBC instance.

F.4 WIDTH/DEPTH SCALING RULES

To study the effect of additional "budget" beyond dataset proxies, we apply scaling rules:

$$S_{\text{model}} = \text{Clamp}\big(\text{Op}_S(S_{\text{base}}; k_S)\big), \qquad L_{\text{model}} = \text{Clamp}\big(\text{Op}_L(L_{\text{base}}; k_L)\big),$$

where Op is either `identity`, `add`, or `mul`, and `Clamp` enforces preset bounds (e.g. $S_{\min} \le S_{\text{model}} \le S_{\max}$). In the main SBC-vs-MLP comparison we use the additive setting

$$S_{\text{model}} = S_{\text{base}} + S_{\text{add}}, \qquad L_{\text{model}} = L_{\text{base}} + L_{\text{add}},$$

with $S_{\text{add}} = 10$ and $L_{\text{add}} = 0$ unless stated otherwise.

F.5 TRAINING SYSTEM USED IN EXPERIMENTS

This section describes the concrete training system used in our experiments. These choices are implementation-level stabilizers for optimizing a differentiable circuit model; they are not required for the proof of universality.

F.5.1 DEPTHSTACK ARCHITECTURE.

Our implementation (`src/ubcircuit/modules.py`) builds a depth-L_{model} stack: (i) optional `BitLifting` mapping $B \mapsto B_{\text{eff}}$, (ii) a first `GeneralLayer` mapping $B_{\text{eff}} \to 2$ wires (with pair selection when $B_{\text{eff}} > 2$), (iii) $L_{\text{model}} - 2$ middle `ReasoningLayers` mapping $2 \to 2$ wires, and (iv) a final `ReasoningLayer` mapping $2 \to 1$ output. Each layer has S_{model} parallel `BooleanUnits` (*heads*), and a row-wise softmax mixer L_ℓ routes head outputs into the next-layer wires. During training, these mixers and the gate and pair selectors use deterministic soft weights. The resulting forward pass is a differentiable relaxation; it is not an exact evaluation of the proof-level random-circuit marginal.

F.5.2 GATE INTERPOLANTS FOR THE 16-GATE HEAD

Our implementation uses a differentiable feature head $\sigma_{16} : [0,1]^2 \to [0,1]^{16}$ that simultaneously relaxes the 16 binary, single-output Boolean gates in Table 3 to inputs in $[0,1]^2$. Both the Lagrange and RBF modes below are differentiable relaxations; they differ in their behavior away from, and at, Boolean corners. For each gate g with truth table vector $Z^{(g)} \in \{0,1\}^4$ ordered by corners $(0,0), (0,1), (1,0), (1,1)$, we define an interpolant

$$\sigma(x \mid Z^{(g)}) \;=\; \sum_{c \in \{00,01,10,11\}} Z_c^{(g)}\, \phi_c(x), \qquad \sigma_{16}(x) \;=\; \big(\sigma(x \mid Z^{(g)})\big)_{g=1}^{16},$$

where $\{\phi_c\}$ is a mode-dependent corner basis. Some modes are exactly corner-interpolating, while others, such as normalized RBF kernels, are smooth relaxations whose sharpness is controlled by a bandwidth parameter.

Mode `lagrange` (multilinear). Let $\bar{x}_{00} = (0,0)$, $\bar{x}_{01} = (0,1)$, $\bar{x}_{10} = (1,0)$, $\bar{x}_{11} = (1,1)$. Define the bilinear Lagrange basis on $[0,1]^2$:

$$\phi_{00}(x) = (1-x_1)(1-x_2), \quad \phi_{01}(x) = (1-x_1)x_2, \quad \phi_{10}(x) = x_1(1-x_2), \quad \phi_{11}(x) = x_1 x_2.$$

This basis satisfies $\phi_{ij}(\bar{x}_{kl}) = \delta_{(ij),(kl)}$ and yields a differentiable multilinear relaxation that agrees exactly with each discrete gate on the four Boolean corners.

Mode `rbf` (normalized Gaussian kernels). Let $w_c(x) = \exp(-\|x - \bar{x}_c\|_2^2/(2s^2))$ for bandwidth $s > 0$ and corners $\bar{x}_c$. Define a partition-of-unity basis by normalization:

$$\phi_c(x) \;=\; \frac{w_c(x)}{\sum_{c'} w_{c'}(x)}.$$

This yields a smooth (C^∞) relaxation whose sharpness is controlled by s. For finite s, the normalized Gaussian basis is not exactly corner-interpolating because all Gaussian weights are positive; as $s \to 0$ it approaches a corner-localized basis, while as $s \to \infty$ the weights become nearly uniform over corners.

Mode `bump` (compactly supported C^∞ kernels). Let $w_c(x) = \psi(\|x - \bar{x}_c\|_2/r)$ where $r > 0$ is a radius and ψ is a standard smooth compactly supported bump with $\psi(t) = 0$ for $t \geq 1$. We again normalize:

$$\phi_c(x) \;=\; \frac{w_c(x)}{\sum_{c'} w_{c'}(x)}.$$

In our code, `sigma16.radius` controls r. When the support radius is chosen so that at each corner $\bar{x}_c$ all other kernels vanish (e.g., $r < 1$, since corner-to-corner distances are ≥ 1), the normalized basis satisfies $\phi_c(\bar{x}_{c'}) = \delta_{c,c'}$.

F.5.3 Layerwise bandwidth schedule.

For `rbf` and the internal bandwidth parameter used by `sigma16`, we optionally anneal the sharpness across layers using a linear schedule

$$s_\ell \;=\; s_{\text{start}} + \frac{\ell}{L_{\text{model}} - 1}\,(s_{\text{end}} - s_{\text{start}}), \qquad \ell = 0, \ldots, L_{\text{model}} - 1.$$

matching the implementation parameters `sigma16.s_start` and `sigma16.s_end`. We report an ablation over `sigma16.mode` in Table 12.

F.5.4 Pair selection and MI-based priors.

For $B_{\text{eff}} > 2$, the first layer uses a differentiable pair selector with two row-softmax matrices $\text{PL}, \text{PR} \in \mathbb{R}^{S_{\text{model}} \times B_{\text{eff}}}$. Their soft weights mix candidate left/right inputs during the relaxed forward pass; $\arg\max$ choices are used only for discrete decoding. When `pair.route=mi_soft`, we compute instance-specific high-mutual-information pairs from the truth table (via `top_mi_pairs` in `src/ubcircuit/mi.py`) and convert them to priors $\text{PL}_{\text{prior}}, \text{PR}_{\text{prior}} \in \Delta_{B_{\text{eff}}}^{S_{\text{model}} \times B_{\text{eff}}}$ (expanded to B_{eff} if lifting is enabled). These are injected as log-probability biases with strength `pair.prior_strength`. When `pair.route=mi_hard`, the MI-selected pairs are used as fixed discrete pairs (`FixedPairSelector`). When `pair.route= learned`, no priors are used.

F.5.5 Repulsive right-pick (optional).

To discourage self-pairs within a head, the pair selector optionally couples the right-pick to the left-pick. If `pair.repel=true`, the implementation supports: (i) log-space repulsion (`pair.mode=log` or `hard-log`), adding $\eta \log(1 - \text{PL})$ to right logits and optionally masking the left argmax coordinate in `hard-log`; and (ii) probability-space repulsion (`pair.mode=mul` or `hard-mul`), multiplying right probabilities by $(1 - \text{PL})$ and optionally hard-zeroing the left argmax coordinate in `hard-mul`. Our best-reported configuration uses MI priors (`pair.route=mi_soft`) with repulsion disabled.

F.5.6 Asynchronous temperature annealing.

Each layer uses a temperature $\tau_\ell(t)$ to control the sharpness of both gate selection and routing. We use asynchronous schedules with `direction` $\in \{$`top_down`, `bottom_up`$\}$ and `schedule` $\in \{$`linear`, `cosine`$\}$. Early in training, larger τ_ℓ keeps selections soft (smoother gradients); later, $\tau_\ell \downarrow T_{\min}$ makes distributions near one-hot, enabling stable $\arg\max$ decoding.

F.5.7 Layerwise σ_{16} bandwidth schedule.

For the `rbf` version of σ_{16}, the parameter $s > 0$ is the Gaussian bandwidth in the corner basis $w_c(x) = \exp(-\|x - \bar{x}_c\|_2^2/(2s^2))$. This bandwidth controls the sharpness of the gate interpolant: smaller s gives a sharper corner-localized basis, while larger s gives a smoother, more blended interpolation. In some runs we use a layer-dependent bandwidth

$$s_\ell = s_{\text{start}} + \frac{\ell}{L_{\text{model}} - 1}\big(s_{\text{end}} - s_{\text{start}}\big), \qquad \ell = 0, \ldots, L_{\text{model}} - 1.$$

This is distinct from the softmax temperature $\tau_\ell(t)$ used for gate and routing distributions.

F.5.8 OBJECTIVE, OPTIMIZER, AND EARLY STOPPING.

All methods are trained on full truth tables using binary cross entropy. SBC optimizes

$$\mathcal{J} = \mathcal{L}_{\text{BCE}} + \mathcal{R}_{\text{ent}} + \mathcal{R}_{\text{div-units}} + \mathcal{R}_{\text{div-rows}} + \mathcal{R}_{\text{const16}}.$$

We use RMSProp in the main runs. Early stopping monitors EM after a warm-up (configurable `min_steps`, `check_every`, `patience_checks`).

All models are trained separately for each task on the full truth table. SBC is optimized through a deterministic differentiable relaxation with soft gate and routing weights; the loss may include entropy/diversity regularizers described below. We use the same early-stopping protocol across methods, monitoring full truth-table EM after a warm-up period. Unless stated otherwise, reported results are mean $\pm$ standard deviation across seeds.

F.5.9 COMPUTE RESOURCES AND SUPPLEMENTAL CODE.

All reported experiments are CPU-based. Each task is trained on its full truth table, so the main memory and time cost scales exponentially in the ambient input dimension B: an instance with dimension B materializes all 2^B assignments and evaluates the model over this table during training and early-stopping checks. In our large runs, each task was launched as an independent CPU job using 4–8 CPU threads and approximately 32–64 GB memory for the main $B \leq 20$ benchmark; exploratory larger-B rows can require more memory because of full-truth-table materialization. Wall-clock time varies substantially with B, model width/depth, and early stopping, ranging from minutes for many MLP rows to several hours for the hardest SBC rows.

We provide an anonymized supplemental code folder containing the core `ubcircuit` package, synthetic dataset-generation scripts, a main configuration file, and local CPU commands for regenerating small smoke tests and the benchmark-generation protocol. Cluster-specific submission scripts are not included in the supplement because they contain institution-specific accounts and file-system paths; the supplied code exposes the model, benchmark-generation procedure, and local training entry points.

F.5.10 REGULARIZERS $\mathcal{R}_\bullet$ USED IN EXPERIMENTS.

In the implementation (`src/ubcircuit/utils.py`, `regularizers_bundle`), the regularizers are computed from the per-layer diagnostics returned by the forward pass. For each layer ℓ, the model exposes: (i) routing rows $L_\ell \in \Delta_S^{B_{\ell+1} \times S}$ (row-softmax over heads), and (ii) head gate logits $\{W^{(\ell:s)}\}_{s=1}^{S}$ (each $W^{(\ell:s)} \in \mathbb{R}^{16}$ when `gate_set=16`). Let τ_ℓ be the layer temperature used at the current optimization step.

Entropy term. The code sums Shannon entropies of each routing row and each head's gate distribution:

$$\mathcal{R}_{\text{ent}} = \lambda_{\text{ent}} \sum_{\ell} \left(\sum_{k=1}^{B_{\ell+1}} H(L_\ell[k,:]) + \sum_{s=1}^{S} H\Big(\text{SM}(W^{(\ell:s)}/\tau_\ell)\Big) \right), \qquad H(q) \triangleq -\sum_i q_i \log q_i.$$

Head-diversity term. Let $p^{(\ell:s)} = \text{SM}(W^{(\ell:s)}/\tau_\ell) \in \Delta_{16}$. The head-diversity penalty is the sum of pairwise cosine similarities:

$$\mathcal{R}_{\text{div-units}} = \lambda_{\text{div-units}} \sum_{\ell} \sum_{1 \leq s < s' \leq S} \cos\Big(p^{(\ell:s)}, p^{(\ell:s')}\Big), \qquad \cos(u,v) = \frac{\langle u, v \rangle}{\|u\|_2 \|v\|_2}.$$

Row-diversity term. The routing diversity penalty is the sum of pairwise cosine similarities between routing rows:

$$\mathcal{R}_{\text{div-rows}} = \lambda_{\text{div-rows}} \sum_{\ell} \sum_{1 \leq k < k' \leq B_{\ell+1}} \cos\Big(L_\ell[k,:], L_\ell[k',:]\Big).$$

Constant-gate penalty (16-gate head). When enabled (`lam_const16`> 0), the code adds a penalty on selecting `FALSE` and `TRUE` on all non-final layers:

$$\mathcal{R}_{\text{const16}} = \lambda_{\text{const16}} \sum_{\ell < L_{\text{model}} - 1} \sum_{s=1}^{S} \Big(p^{(\ell:s)}_{\texttt{FALSE}} + p^{(\ell:s)}_{\texttt{TRUE}} \Big),$$

Component	Proof-level model	Legacy experimental implementation
Layer topology	A complete layered state with widths $B_0, \ldots, B_\Delta$ is preserved. Each of the B_{l+1} neurons may select two parents from all B_l preceding outputs.	Each layer evaluates S_{model} candidate heads, after which a mixer retains only two wires between layers and one wire at the output. Subsequent heads share the retained pair.
Input lifting	The stochastic bit-lifting channel $\mathbf{X}^{\uparrow:W}$ is part of the construction (Section 3.1).	`BitLifting` is implemented, but the best-performing reported setting uses `lifting.use=false`.
Gate semantics	Each neuron samples one of the 16 binary, single-output gates, producing a discrete circuit for every outcome ω (table 3 and theorem 1).	Lagrange or RBF gate interpolants and soft gate weights define a deterministic differentiable relaxation during training; local $\arg\max$ selects a discrete gate at decoding (sections F.5.2 and F.5.11).
Edge selection	Every non-input neuron has two selectors over the full preceding proof-level layer; sampled selectors define discrete edges (section 3.2.1).	The first layer uses `PairSelector`; later heads read the shared two-wire state. Soft routing and mixers are differentiable relaxations, with optional MI priors and repulsion (section F.5.4).
Predictive quantity	$p_\theta(x) = \mathbb{E}_{\mathcal{C}\sim\mu_\theta}[f_{\mathcal{C}}(x)]$ is the exact random-circuit marginal.	$\hat{p}_\theta(x)$ is produced by deterministic soft propagation through the restricted stack. It is not generally equal to the exact random-circuit marginal (section F.2).
Decoding	Circuit realization is defined by jointly sampling an outcome ω (theorem 1).	Post-hoc local $\arg\max$ choices produce one symbolic circuit and gate histograms (sections F.5.11 and F.8).
Optimization aids	Not required for the universality and existence proofs.	Asynchronous temperature annealing and regularizers stabilize the relaxed optimization and encourage crisp decoding (section F.5.8).

Table 4: Distinction between the proof-level architecture and the restricted differentiable training system used in the reported experiments.

and uses $\mathcal{R}_{\text{const16}} = 0$ on the final layer. (In the best-reported setting we set $\lambda_{\text{const16}} = 0$.)

F.5.11 Decoding and evaluation.

Decoding is post-hoc: after training we select gates and routing via $\arg\max$ at each layer to obtain a discrete circuit and a symbolic expression. EM is computed from the model's truth-table predictions (threshold at 0.5) and is independent of symbolic string equality.

F.6 Proof-Level Construction vs. Experimental Instantiation

The proof-level architecture and the legacy experimental implementation do not have the same layer topology. The theory preserves a complete layered state: every neuron in layer $l+1$ may select two parents from all B_l outputs of layer l, and all B_{l+1} outputs are retained for the next layer. By contrast, the experimental `DepthStack` computes S_{model} candidate heads at each layer and then compresses them to two retained wires between layers (and one wire at the output). All heads in the next layer therefore receive the same two-wire state. Consequently, the universality and size theorems apply to the proof-level architecture, not to this restricted legacy topology. A hard-decoded `DepthStack` is still a valid circuit of binary, single-output gates, but its empirical results should be interpreted as results for that restricted differentiable training system. Table 4 records the distinction explicitly.

F.7 Interpretability metrics for continuous baselines

BNR (Appendix B.1) is defined for Boolean-valued maps. MLP hidden units are real-valued, so we compute operational BNR diagnostics and additional gate-recoverability probes by enumerating each unit on the full truth table.

BNR diagnostics (rounded and tolerant). Let $u : \{0,1\}^B \to \mathbb{R}$ be a hidden unit evaluated on all 2^B inputs. We report two operational variants:

- **BNR$_{\text{exact}}$ (rounded).** Fix a rounding precision d (we use $d = 6$). Declare u BNR if
$$\#\big\{\text{Round}_d(u(x)) : x \in \{0,1\}^B\big\} \le 2.$$
- **BNR$_\varepsilon$ (two-cluster tolerance).** Fix $\varepsilon > 0$ (we use $\varepsilon = 10^{-3}$). Split the multiset $\{u(x)\}$ at its median, take the median of each half as centers c_0, c_1, and declare u ε-BNR if
$$\max_{x \in \{0,1\}^B} \min\big\{|u(x) - c_0|,\, |u(x) - c_1|\big\} \le \varepsilon.$$

For a layer with H units we report the fraction of units satisfying each criterion. For an L-layer MLP we report both the first-layer value (L1) and the average across hidden layers (denoted "all"). SBC is Boolean by construction, hence achieves BNR$= 1$ for all internal units.

Deterministic binarization for gate probes. To probe Boolean decision patterns of MLP units, we map a real unit to a Boolean function by thresholding at the all-zero assignment:
$$t(u) \triangleq u(\mathbf{0}), \qquad \hat{u}(x) \triangleq \mathbb{I}\{u(x) \ge t(u)\} \in \{0,1\}.$$
This binarization is used only for the gate-recoverability probes below. Because the threshold is non-strict, any unit that is constant on the Boolean cube is mapped to TRUE: $u(x) = t(u)$ implies $\hat{u}(x) = 1$ for every x. Thus, dead or constant ReLU units contribute directly to the TRUE peak in Figure 3.

Primitive recoverability on the first hidden layer. For each first-layer binarized unit $\hat{u}$, define
$$\text{bestAcc}_{\text{in}}(u) \triangleq \max_{p \in \mathcal{P}_{\text{in}}} \Pr_{x \sim \text{Unif}(\{0,1\}^B)} \big[\hat{u}(x) = p(x)\big].$$
Here $\mathcal{P}_{\text{in}}$ is a set of Boolean functions $p : \{0,1\}^B \to \{0,1\}$:
$$\mathcal{P}_{\text{in}} \stackrel{\text{def}}{=} \{x \mapsto x_i,\ x \mapsto 1 - x_i : i \in [B]\} \cup \{x \mapsto g(x_i, x_j) : g \in \mathbb{G}^\star,\ i, j \in [B],\ i \ne j\}.$$
Thus $p(x)$ is well-defined for every $p \in \mathcal{P}_{\text{in}}$ and $x \in \{0,1\}^B$. We report
$$\text{primHit}_{\text{in}} \triangleq \Pr_u[\text{bestAcc}_{\text{in}}(u) = 1], \qquad \text{primBest}_{\text{in}} \triangleq \mathbb{E}_u[\text{bestAcc}_{\text{in}}(u)].$$
Here **primHit** is an exact recovery rate, while **primBest** averages the best-match accuracies.

Layerwise primitive recoverability (all layers). Let $\hat{z}_1^{(\ell-1)}, \ldots, \hat{z}_{H_{\ell-1}}^{(\ell-1)}$ denote the binarized units in layer $\ell - 1$. For a binarized unit $\hat{u}_j^{(\ell)}$ in layer ℓ, define
$$\text{bestAcc}_{\text{layer}}^{(\ell)}(u_j) \triangleq \max_{p \in \mathcal{P}_{\text{layer}}^{(\ell)}} \Pr_{x \sim \text{Unif}(\{0,1\}^B)} \big[\hat{u}_j^{(\ell)}(x) = p(x)\big],$$
where $\mathcal{P}_{\text{layer}}^{(\ell)}$ is the set of Boolean functions $p : \{0,1\}^B \to \{0,1\}$ of the form
$$p(x) = g\Big(\hat{z}_i^{(\ell-1)}(x), \hat{z}_k^{(\ell-1)}(x)\Big), \qquad g \in \mathbb{G}^\star, \quad i \ne k.$$
We define per-layer summaries
$$\text{primHit}_{\text{layer}}^{(\ell)} \triangleq \Pr_j[\text{bestAcc}_{\text{layer}}^{(\ell)}(u_j) = 1], \qquad \text{primBest}_{\text{layer}}^{(\ell)} \triangleq \mathbb{E}_j[\text{bestAcc}_{\text{layer}}^{(\ell)}(u_j)],$$
and report their averages across hidden layers (denoted "all").

Table 5: Exact-match (EM) on truth-table evaluation (mean $\pm$ std across 5 seeds). SBC EM is reported once (canonical SBC setting) and MLP EM is shown for each matching regime.

MLP match	Avg. MLP params	MLP EM	SBC EM
neuron	405	0.957 ± 0.002	0.961 ± 0.003
param	758	0.977 ± 0.007	

Table 6: BNR diagnostics (mean $\pm$ std across 5 seeds). "all" denotes the average across hidden layers. SBC is BNR by construction, hence BNR$\equiv 1$.

Model / match	$\text{BNR}_{\text{exact}}$(L1)	$\text{BNR}_{\text{exact}}$(all)	BNR_{ε}(L1)	BNR_{ε}(all)
SBC	1.000	1.000	1.000	1.000
neuron	0.114 ± 0.003	0.209 ± 0.002	0.115 ± 0.004	0.209 ± 0.002
param	0.107 ± 0.002	0.199 ± 0.001	0.107 ± 0.002	0.199 ± 0.001

F.8 Gate histogram diagnostics

We record gate-type histograms as descriptive diagnostics:

- **SBC-all:** gate histogram across all units and layers under $\arg\max$ decoding.
- **SBC-path:** gate histogram restricted to the decoded readout path.
- **MLP-all:** histogram of exact gate matches from the layerwise recoverability probe, aggregated across layers.

These histograms are not correctness certificates; they help interpret which primitives dominate the decoded computation. (In our runs, MLP exact-match recoveries are often dominated by constant-type behaviors such as `TRUE`, so we interpret high **primBest** jointly with **primHit** and the gate histograms.)

G Additional Results and Diagnostics

G.1 EM and BNR

We summarize (i) task correctness via exact match (EM) and (ii) neuronwise Boolean-valuedness via BNR. EM is computed on the full truth table for each instance (Section F.1); BNR is defined in Appendix F.7. Table 5 reports EM for MLP baselines under each matching regime, while reporting the SBC EM once under the fixed canonical SBC setting used throughout. Table 6 reports BNR diagnostics: for SBC BNR$\equiv 1$ holds by construction (Theorem 1); for MLPs, BNR is measured post-hoc on hidden activations.

Takeaway. Across matching regimes, MLP achieves higher EM than SBC (Table 5), while SBC guarantees neuronwise Boolean-valuedness (Table 6). For MLPs, strict two-valuedness is rare, especially at early layers; the tolerance-based BNR_{ε} closely tracks the rounded $\text{BNR}_{\text{exact}}$ in this regime.

G.2 Performance by Ambient Input Dimension

Figure 2 breaks down truth-table EM by ambient input dimension on the same benchmark summarized in Table 1. This diagnostic shows that SBC is especially competitive at larger input dimensions, while the aggregate table reports performance over the full $B = 2, \ldots, 20$ range.

G.3 MAP Decoding Diagnostics

The main experiments evaluate SBC through the deterministic differentiable relaxation described in Appendix F.2. Extracting a single discrete circuit from its learned soft gate and routing weights is a related but separate problem, closer in spirit to discrete program extraction or combinatorial

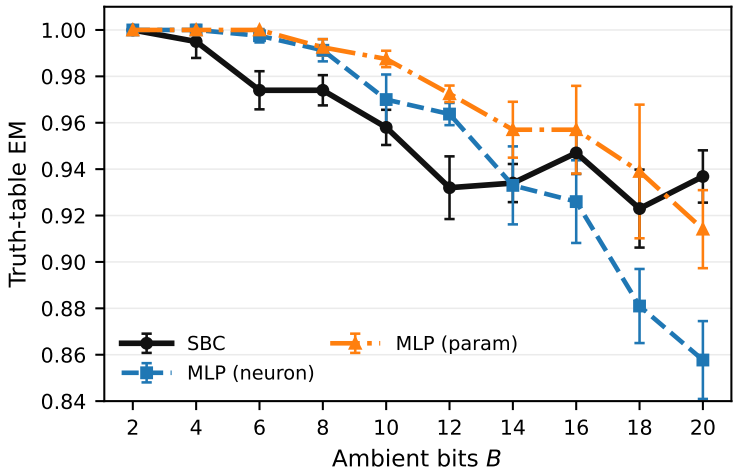


Figure 2: Truth-table EM by ambient input dimension B on the full benchmark. This is a diagnostic breakdown of the same tasks summarized in Table 1, not a separately selected evaluation subset.

structure search. The theoretical claims concern the support and universality of the proof-level random-circuit model, whereas this diagnostic compares the experimental relaxation with local deterministic decoding.

Given a trained model, we apply a simple local $\arg\max$ decoder that selects the largest gate and routing weight at each choice. This produces one discrete circuit, but it does not optimize globally over deterministic circuits or directly minimize truth-table error.

Table 7: Relaxed prediction versus local $\arg\max$ decoding for an auxiliary decode-oriented SBC setting (mean $\pm$ std across 3 seeds). The gap illustrates that an accurate differentiable relaxation need not yield a single equally accurate circuit under independent local choices.

Setting	Relaxed EM	Locally decoded EM	Gap
Decode-oriented SBC	0.771 ± 0.003	0.248 ± 0.002	0.523 ± 0.004

This diagnostic is consistent with the scope of the paper. Optimizing a relaxed object and extracting one globally accurate discrete object are distinct algorithmic problems, a distinction that also appears in program synthesis, planning, discrete structure learning, and combinatorial optimization. Developing decoders or training objectives that directly favor an accurate decoded circuit is an interesting direction for future work.

G.4 Primitive recoverability

BNR measures whether hidden activations are strictly two-valued; it does not ask whether a unit behaves like a *specific* Boolean gate. To probe gate-like *decision patterns* in MLPs, we report fan-in-2 primitive recoverability after deterministic binarization (Appendix F.7). Table 8 summarizes recoverability both (i) against primitives over *input bits* (L1) and (ii) layerwise relative to binarized previous-layer units (all).

Takeaway. Primitive recoverability can be substantially higher than strict BNR (compare Tables 6 and 8), indicating that many MLP units induce gate-like *Boolean patterns* after binarization even when their real-valued magnitudes remain multi-level.

G.5 Gate histograms (SBC vs. MLP)

To contextualize Table 8, we visualize which gate types dominate the recovered computations. Figure 3 compares: (i) SBC-all (all decoded units), (ii) SBC-path (decoded readout path), and (iii)

Table 8: Fan-in-2 primitive recoverability after deterministic binarization (mean ± std across 5 seeds). primHit is the fraction of units with an exact primitive match; primBest is the average best-match accuracy. "in" evaluates first-layer units against primitives over input bits; "layer" evaluates gate recoverability relative to binarized previous-layer units, averaged across hidden layers.

MLP match	$\text{primHit}_{\text{in}}$(L1)	$\text{primBest}_{\text{in}}$(L1)	$\text{primHit}_{\text{layer}}$(all)	$\text{primBest}_{\text{layer}}$(all)
neuron	0.463 ± 0.004	0.945 ± 0.001	0.552 ± 0.002	0.953 ± 0.000
param	0.466 ± 0.003	0.941 ± 0.000	0.559 ± 0.002	0.953 ± 0.000

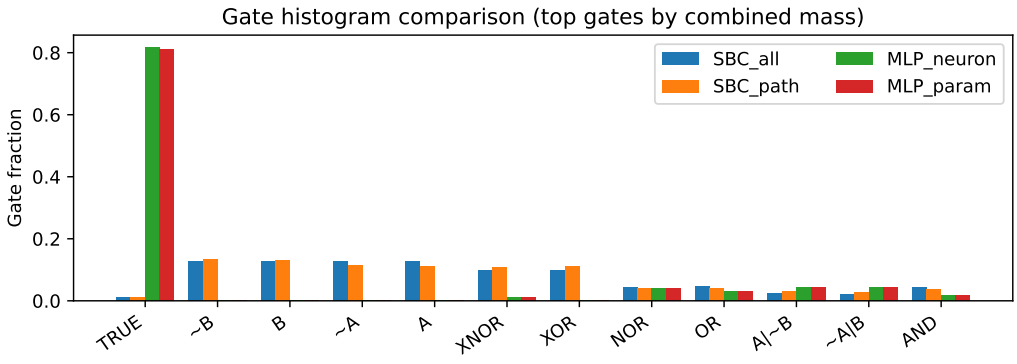


Figure 3: Gate histograms: SBC-all, SBC-path, and MLP-all (exact recovered gates) across matching regimes.

MLP-neuron and MLP-param (exact recovered gates from the layerwise primitive probe, aggregated across layers) for each matching regime (Appendix F.8).

Takeaway. Although Table 8 reports high `primBest` values, Figure 3 reveals that the MLP exact-hit histogram is heavily dominated by the constant `TRUE` gate in this regime. Thus, primitive recoverability should be interpreted jointly with the gate histogram: a high best-match score can reflect substantial mass on trivial (constant) gates rather than widespread recovery of non-constant fan-in-2 logic. This motivates treating BNR (Table 6) as the primary neuronwise interpretability certificate, with primitive probes serving as auxiliary diagnostics.

G.6 Expression complexity

We additionally report a coarse expression-complexity proxy based on token counts (number of variable occurrences plus Boolean operators) of the *label* formulas and the *decoded SBC* expressions. Since dataset formulas are generated without algebraic simplification, shorter decoded expressions should be interpreted only as a simplicity bias among logically equivalent representations, not as a semantic advantage (Table 9).

G.7 How we selected the final hyperparameter setting (ablation path)

We selected the final experimental configuration via a staged ablation, keeping the dataset and evaluation protocol fixed (full truth-table training and EM evaluation). The stages are:

Stage I: coarse hyperparameter grid (including lifting). We first ran a broad grid over routing strategy, repulsion settings, optimizer choice, and the lifting toggle (among other knobs), and ranked configurations by EM. Table 10 and Table 11 reports the top configurations (aggregated across seeds), and motivates our choice of the baseline hyperparameter setting used in later stages. In this stage, lifting (`lifting.use`) is treated as a tunable option.

Stage II: 16-gate interpolant ablation (holding the Stage-I setting fixed). Fixing the best-performing Stage-I hyperparameters, we ablated the differentiable 16-gate interpolant used during

Table 9: Average token complexity of label expressions vs. decoded SBC expressions (mean ± std across seeds). Label expressions are generated without algebraic simplification; decoded SBC expressions are often shorter among logically equivalent forms.

Setting	label tok	SBC tok
SBC (main)	11.80 ± 0.00	9.80 ± 0.15

Table 10: Top-8 configurations on full_grid (mean ± std across seeds).

route	repel	mode	eta	lam_const16	EM	n
mi_soft	False	log	2	0	0.8651 ± 0.0124	20
mi_soft	False	log	2	0.005	0.8578 ± 0.0126	20
mi_soft	False	log	2	0.001	0.8557 ± 0.0126	20
mi_soft	True	hard-log	2	0.001	0.8435 ± 0.0006	19
mi_soft	True	log	2	0.001	0.8435 ± 0.0006	20
mi_soft	True	log	2	0.005	0.8435 ± 0.0004	18
mi_soft	True	hard-log	2	0.005	0.8435 ± 0.0004	20
mi_soft	True	hard-log	2	0	0.8432 ± 0.0012	20

training: `sigma16.mode` $\in \{$`rbf`, `bump`, `lagrange`$\}$. Table 12 summarizes the results and shows that `rbf` performs best in our regime; we therefore fix `sigma16.mode=rbf` for subsequent experiments.

We selected the reference SBC budget using an auxiliary width/depth sweep on a computationally lighter subset of the same generator family, then fixed $S_{\text{model}} = S_{\text{base}} + 10$ and $L_{\text{model}} = L_{\text{base}}$ for all main comparisons.

H Additional Related Work

Differentiable logic gate networks. Petersen et al. (2022) learn softmax distributions over binary Boolean operators and discretize by selecting the most likely gate. Their connectivity is fixed after random initialization, whereas SBC learns stochastic routing and bit-lifting in addition to gates. Thus our model defines a distribution over circuit topologies and operators, which is central to our circuit-validity and universality results.

Neuro-symbolic reasoning and probabilistic circuits. Neuro-symbolic methods integrate neural components with symbolic structure or logic-based inference. Logic Tensor Networks enforce differentiable satisfaction of first-order constraints during training Badreddine et al. (2022), while neural probabilistic logic programming systems such as DeepProbLog embed neural predicates inside probabilistic programs for joint learning and inference Manhaeve et al. (2018). Probabilistic circuits, including sum-product networks (SPNs), represent distributions as structured compositions that admit efficient exact inference under tractable structural conditions Poon & Domingos (2011); Choi et al. (2020). Our approach is closest in spirit in that it builds in an intrinsic circuit-level representation rather than extracting one post hoc; we specialize to circuits of binary, single-output Boolean gates and obtain an explicit certificate of neuronwise Booleanity. Each gate output may be reused downstream. Extending the construction to richer circuit families (e.g., variable-arity or multi-output gates, or alternative gate sets) is a natural direction for future work.

Table 11: Top-8 configurations on lift_full_grid (mean ± std across seeds).

route	repel	mode	eta	lam_const16	EM	n
learned	True	hard-log	2	0.005	0.7496 ± 0.0056	20
learned	True	hard-log	2	0	0.7457 ± 0.0055	18
learned	True	hard-log	2	0.001	0.7408 ± 0.0078	19
learned	True	log	2	0.005	0.7391 ± 0.0045	20
learned	True	log	2	0	0.7367 ± 0.0061	18
learned	False	log	2	0	0.7366 ± 0.0055	18
learned	False	log	2	0.005	0.7357 ± 0.0044	15
learned	False	log	2	0.001	0.7296 ± 0.0079	18

Table 12: Ablation over the σ_{16} gate-interpolant choice on the `mode_cmp` suite (mean ± std across seeds).

σ_{16} mode	EM	row-acc
rbf	0.8721 ± 0.0024	0.9908 ± 0.0004
bump	0.8477 ± 0.0044	0.9873 ± 0.0006
lagrange	0.5694 ± 0.0178	0.9552 ± 0.0022