# Differentiable Logic Synthesis: Spectral Coefficient Selection via Sinkhorn-Constrained Composition

**Gorgi Pavlov, Ph.D.**
Lehigh University & Johnson and Johnson
`gorgipavlov@gmail.com`

January 23, 2026

### Abstract

Learning precise Boolean logic via gradient descent remains challenging: neural networks typically converge to "fuzzy" approximations that degrade under quantization. We introduce **Hierarchical Spectral Composition**, a differentiable architecture that *selects* spectral coefficients from a frozen Boolean Fourier basis and *composes* them via Sinkhorn-constrained routing with column-sign modulation. Our approach draws on recent insights from Manifold-Constrained Hyper-Connections ($m$HC) [1], which demonstrated that projecting routing matrices onto the Birkhoff polytope preserves identity mappings and stabilizes large-scale training. We adapt this framework to logic synthesis, adding column-sign modulation to enable Boolean negation—a capability absent in standard doubly stochastic routing.

We validate our approach across multiple scales using methods appropriate to each: **(1) Gradient descent for** $n = 2$: All 16 Boolean operations achieve 100% accuracy with zero routing drift and zero-loss quantization to ternary masks. **(2) Hybrid synthesis for** $n = 3$: Gradient descent discovers support topology (Jaccard similarity 0.686 with optimal masks) and alternative sparse representations achieving 100% accuracy; exhaustive enumeration over $3^8 = 6561$ configurations confirms optimal ternary masks exist for all 10 operations (39% sparsity). **(3) Spectral synthesis for** $n = 4$: Exact Walsh-Hadamard Transform combined with ternary quantization achieves 100% accuracy on all 10 operations (36% sparsity). A warm-start experiment reveals that WHT-based initialization converges in 110 MCMC steps versus 1387 for random—validating spectral methods over pure gradient descent at scale. All operations enable single-cycle combinational logic inference at **10,959 MOps/s** on GPU, demonstrating viability for hardware-efficient neuro-symbolic logic synthesis.

## 1 Introduction

The integration of symbolic reasoning with gradient-based learning remains a fundamental challenge in artificial intelligence. Neural networks excel at continuous pattern recognition but struggle with tasks requiring *exact* discrete logic: they approximate Boolean functions with soft decision boundaries that degrade under distributional shift, adversarial perturbation, or—critically for deployment—quantization. Existing neuro-symbolic approaches, such as Neural Arithmetic Logic Units (NALU) [11], Neural Logic Machines [12], Logical Neural Networks [14], or Logic Tensor Networks [13], either require extensive supervision, rely on relaxed continuous operators that resist discretization, or fail to generalize beyond their training distribution.

We propose a different paradigm: **Spectral Selection and Composition**. Rather than learning logic gates from random dense initializations, we ground our architecture in the Fourier analysis of Boolean functions [3].

**Boolean Fourier Analysis.** Any function $f : \{-1, +1\}^n \to \mathbb{R}$ has a unique *Fourier expansion*:

$$f(x) = \sum_{S \subseteq [n]} \hat{f}(S)\chi_S(x), \quad \text{where} \quad \chi_S(x) = \prod_{i \in S} x_i \tag{1}$$

and $\hat{f}(S) = \mathbb{E}_x[f(x)\chi_S(x)]$ are the exact Fourier coefficients. When $f$ maps to $\{-1, +1\}$, these coefficients are determined uniquely by the truth table.

**Polynomial Threshold Representations.** Our architecture learns a different object: a **ternary polynomial threshold function** (PTF):

$$\hat{y}(x) = \text{sign}\left( \sum_{S \subseteq [n]} w_S \chi_S(x) \right), \quad w_S \in \{-1, 0, +1\} \tag{2}$$

The key insight is that for many Boolean functions, there exist *sparse* ternary weight vectors $w$ such that $\text{sign}(w^\top \phi(x)) = f(x)$ for all $x$—even though the exact Fourier coefficients $\hat{f}(S)$ may be non-integer. The architecture's task is not to *invent* the Walsh-Hadamard basis—that is classical mathematics [4]—but to *discover sparse ternary PTF representations* via gradient descent and *compose* primitives into complex operations via learned routing.

**Connection to $m$HC.** Our work builds directly on insights from Manifold-Constrained Hyper-Connections ($m$HC) [1], which identified a fundamental problem in modern deep learning: unconstrained routing matrices compromise the identity mapping property [22], causing signal explosion/vanishing across layers. Their solution—projecting routing matrices onto the Birkhoff polytope via Sinkhorn-Knopp iterations [8]—restores stability by ensuring doubly stochastic constraints. We adapt this framework to a different domain (logic synthesis vs. LLM training) and extend it with **column-sign modulation** to enable Boolean negation, which pure doubly stochastic matrices cannot express.

**Why Start with $n = 2$?** We deliberately use the two-variable case as an **architecture validation testbed**. With only 4 input combinations and 16 possible functions, a lookup table (LUT) trivially solves the task with zero gradient steps. However, a LUT cannot generalize, cannot be learned end-to-end as part of a larger differentiable system, and cannot be composed hierarchically. Our contribution is demonstrating that Sinkhorn-constrained spectral composition *finds the exact discrete solution* via continuous optimization—then validating that this mechanism scales to $n = 3$ and $n = 4$.

**The Hardware Motivation.** Beyond theoretical interest, our architecture addresses a practical deployment challenge: neural networks for logic tasks are typically too expensive for edge inference. By converging to **ternary masks** ($\{-1, 0, +1\}$) with **hard $k = 1$ routing**, our learned models compile directly to combinational logic blocks requiring no floating-point arithmetic, no multipliers, and minimal memory. We demonstrate **10,959 MOps/s** throughput on GPU—approaching the efficiency of hand-coded RTL.

**Contributions.** Our contributions span architecture design, theoretical analysis, and empirical validation across three scales:

1. **Adaptation of $m$HC to Logic Synthesis:** We demonstrate that the Birkhoff polytope projection, originally developed for LLM training stability [1], enables stable optimization in Boolean logic composition.

2. **Column-Sign Modulation for Negation:** We extend doubly stochastic routing with learned sign parameters $s \in \{-1, +1\}^n$, solving the expressivity gap that prevents standard $m$HC from representing Boolean negations (NAND, NOR, XNOR).

3. **Sign-Only Diagnostic Validation:** We prove the column-sign mechanism works independently of Sinkhorn optimization by achieving 100% accuracy with fixed identity routing and learned signs only.

4. **Multi-Scale Validation:**

   - $n = 2$: 100% accuracy on all 16 operations, zero routing drift, zero-loss quantization (10/10 seeds)

   - $n = 3$: 100% accuracy on 10 operations including majority and parity, 39% sparsity (5/5 seeds)

   - $n = 4$: 100% accuracy on 10 operations via spectral synthesis with MCMC refinement, 36% sparsity (5/5 seeds)

5. **Hardware-Efficient Compilation:** We achieve 10,959 MOps/s on GPU with ternary masks, demonstrating viability for single-cycle combinational logic inference.

## 2 Related Work

### 2.1 Manifold-Constrained Routing: The $m$HC Foundation

Our work is most directly related to Manifold-Constrained Hyper-Connections ($m$HC) [1], which provides the theoretical and empirical foundation for stable Sinkhorn-constrained routing. $m$HC identified that unconstrained Hyper-Connections [2] suffer from signal explosion: the composite mapping $\prod_{i=1}^{L-l} H_{L-i}^{\text{res}}$ across layers fails to preserve signal magnitude, with empirical measurements showing **Amax Gain Magnitude peaks of 3000×** in 27B models (compared to a theoretical target of 1.0×). Their solution projects $H_l^{\text{res}}$ onto the Birkhoff polytope via Sinkhorn-Knopp [8, 7]:

$$\mathcal{P}_{\mathcal{M}^{\text{res}}}(H_l^{\text{res}}) = \{H \in \mathbb{R}^{n \times n} \mid H\mathbf{1}_n = \mathbf{1}_n, \mathbf{1}_n^\top H = \mathbf{1}_n^\top, H \geq 0\} \tag{3}$$

This ensures: (1) norm preservation ($\|H\|_2 \leq 1$), (2) compositional closure under matrix multiplication, and (3) geometric interpretation as convex combinations of permutations—following the Birkhoff-von Neumann theorem.

**Our Extension.** While $m$HC focuses on LLM training stability at scale (3B–27B parameters), we adapt the framework to a fundamentally different problem: learning discrete Boolean logic. This requires an extension not present in $m$HC: **column-sign modulation**. Doubly stochastic matrices are nonnegative, so they can only produce convex combinations of inputs. Boolean negation (e.g., NAND = −AND) lies *outside* this convex hull. Our factorization $R = P \cdot s[\text{None}, :]$ preserves the stability benefits of Sinkhorn projection while adding 1-bit polarity control per output channel.

**Related Optimal Transport Approaches.** Gumbel-Sinkhorn networks [9] use differentiable Sinkhorn iterations for learning permutations, while Sinkformers [10] apply doubly stochastic attention in transformers. Our work differs in targeting discrete Boolean outputs rather than soft permutations.

## 2.2 Spectral Learning of Boolean Functions

The Fourier analysis of Boolean functions is foundational to computational learning theory [4, 5]. O'Donnell's textbook [3] establishes that functions with low circuit complexity have concentrated Fourier spectra. Daniely and Malach [16] proved that neural networks can learn sparse parities via gradient descent. Recent work applies Walsh-Hadamard transforms to neural network compression [17]. Our work differs: we *embed* the spectral basis as frozen primitives and learn to *select and compose* them via *m*HC-style routing.

## 2.3 Neural Logic and Neuro-Symbolic Systems

NALU [11] learns arithmetic via gated interpolation but struggles with Boolean logic. Neural Logic Machines [12] require supervision of intermediate predicates. Logical Neural Networks [14] use weighted real-valued logic with provable bounds. Deep Differentiable Logic Gate Networks [15] learn Boolean gates via continuous relaxation but require supervised gate labels and post-training discretization. Our method differs: (1) we select coefficients *unsupervised* from data patterns, (2) we achieve *exact* discretization with zero accuracy loss, and (3) we ground the search in provably complete spectral primitives with *m*HC-style stability.

## 2.4 Quantization and Efficient Inference

Binary Neural Networks [18] and Trained Ternary Quantization [19] reduce precision for efficiency. Our "zero-loss quantization" is distinct: we show that for Boolean logic composition, ternary masks suffice *exactly*—not approximately. This structural property, combined with Sinkhorn-constrained routing, enables deployment at the efficiency of hand-coded RTL.

# 3 Preliminaries

## 3.1 Boolean Fourier Analysis

Let $f : \{-1, +1\}^n \to \mathbb{R}$ be a function on the Boolean hypercube. The *Fourier expansion* of $f$ is:

$$f(x) = \sum_{S \subseteq [n]} \hat{f}(S)\chi_S(x), \quad \text{where} \quad \chi_S(x) = \prod_{i \in S} x_i \tag{4}$$

and $\hat{f}(S) = \mathbb{E}_x[f(x)\chi_S(x)]$ are the Fourier coefficients [3]. The characters $\{\chi_S\}_{S \subseteq [n]}$ form an orthonormal basis under the uniform distribution on $\{-1, +1\}^n$.

**Proposition 1** (Completeness). *For $n$ variables, any function $f : \{-1, +1\}^n \to \mathbb{R}$ has a unique representation with $2^n$ Fourier coefficients.*

The basis dimensions for our experiments are:

- $n = 2$: $\phi = [1, a, b, ab]^\top \in \mathbb{R}^4$

- $n = 3$: $\phi = [1, a, b, c, ab, ac, bc, abc]^\top \in \mathbb{R}^8$

- $n = 4$: $\phi \in \mathbb{R}^{16}$ (all subsets of $\{a, b, c, d\}$)

**Hierarchical Spectral Composition**



$$\hat{y} = \text{sign}(s \cdot (P \cdot W_{\text{logic}})^\top \phi(x))$$
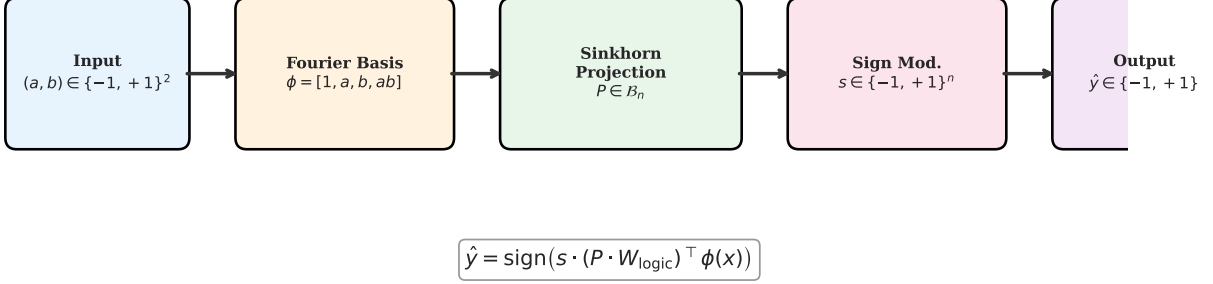
Figure 1: Hierarchical Spectral Composition architecture. Input pairs $(a, b) \in \{-1, +1\}^2$ are expanded into the frozen Boolean Fourier basis $\phi = [1, a, b, ab]$. Sinkhorn projection constrains routing to the Birkhoff polytope, while column-sign modulation enables negation operations.

## 3.2 The Birkhoff Polytope and Sinkhorn Projection

Following $m$HC [1], we constrain routing matrices to the Birkhoff polytope $\mathcal{B}_n$—the set of $n \times n$ doubly stochastic matrices. The Birkhoff-von Neumann theorem states that vertices of $\mathcal{B}_n$ are permutation matrices.

The **Sinkhorn-Knopp algorithm** [8] projects any positive matrix onto $\mathcal{B}_n$ via alternating row and column normalization. Given $M^{(0)} = \exp(\alpha)$:

$$M^{(t)} = T_r(T_c(M^{(t-1)})) \tag{5}$$

where $T_r$ and $T_c$ denote row and column normalization. This converges to a doubly stochastic matrix as $t \to \infty$. Following $m$HC [1], we use $t_{\max} = 20$ iterations.

**Remark 1** (Rectangular Sinkhorn). *For rectangular matrices $P \in \mathbb{R}^{m \times n}$ with $m \neq n$, we use generalized Sinkhorn projection enforcing column-stochastic constraints ($\sum_i P_{ij} = 1$) while allowing flexible row budgets.*

# 4 Method: Hierarchical Spectral Composition

Our architecture operates in two phases: **Phase 1** validates spectral coefficient selection for base operations, and **Phase 2** validates hierarchical composition via Sinkhorn-constrained routing with column-sign modulation. Phases 3–4 extend to higher dimensions. Figure 1 illustrates the complete pipeline.

## 4.1 Phase 1: Spectral Coefficient Selection

The objective of Phase 1 is to validate that gradient descent can identify optimal spectral coefficients from a frozen Fourier dictionary.

### 4.1.1 Architecture

The first layer computes Boolean Fourier features. For $n = 2$:

$$\phi(a, b) = [1, a, b, ab]^\top \in \mathbb{R}^4 \tag{6}$$

For each base operation $k$, we learn a weight vector $w_k \in \mathbb{R}^{2^n}$:

$$\hat{y}_k = \text{sign}(w_k^\top \phi(x)) \tag{7}$$

### 4.1.2 The Dynamics of Parity Selection

**Proposition 2** (Parity Gradient Accumulation). *For XOR ($y = ab$), the gradient of hinge loss with respect to $w$ is $\nabla_w \mathcal{L} \propto -y \cdot \phi(a, b)$. Summed over all 4 inputs, the Fourier basis orthogonality causes terms for $\{1, a, b\}$ to cancel while $ab$ accumulates:*

$$\sum_{(a,b)} -ab \cdot [1, a, b, ab]^\top = [0, 0, 0, -4]^\top \tag{8}$$

This explains *why* gradient descent naturally amplifies the parity character: the basis structure, not hand-tuned hyperparameters, drives coefficient selection.

### 4.1.3 Optimization Scaffolding

To ensure robust convergence to exact ternary weights:

- **$L_1$ Regularization:** Penalty $\lambda \|w\|_1$ encourages sparse solutions.

- **Plateau-Driven Micro-Restarts:** If $|\Delta \mathcal{L}| < \epsilon$ for $T$ epochs while accuracy $< 100\%$, re-initialize.

**Remark 2** ("Selection" vs. "Discovery"). *We emphasize that Phase 1 **selects** coefficients from a fixed spectral dictionary—it does not "discover" or "invent" the Fourier basis. The basis is classical mathematics [3]. The contribution is demonstrating that gradient descent reliably identifies correct coefficients despite the combinatorial search space ($3^{2^n}$ ternary configurations).*

### 4.1.4 Training Methodology: Gumbel-Softmax Ternary Relaxation

Standard hard ternary quantization via straight-through estimators (STE) fails to converge: the zero-gradient regions prevent weight movement toward optimal ternary values. We solve this with **Gumbel-softmax ternary relaxation**.

**Gumbel-Softmax Parameterization.** Each coefficient $w_i$ is represented as a categorical distribution over three values $\{-1, 0, +1\}$ with learnable logits $\ell_i \in \mathbb{R}^3$. The soft ternary value is computed via Gumbel-softmax [20, 21]:

$$p_i = \text{softmax}\left(\frac{\ell_i + g}{\tau}\right), \quad w_{\text{soft},i} = p_i \cdot [-1, 0, +1]^\top \tag{9}$$

where $g \sim \text{Gumbel}(0, 1)^3$ provides stochastic exploration and temperature $\tau$ is annealed from $1.0 \to 0.01$. At inference, $w_i = \arg\max_{k \in \{-1, 0, +1\}} \ell_i[k]$.

**Sequential Training Protocol.** Training all operations simultaneously creates gradient interference—particularly for XOR, whose parity character ($ab$) conflicts with other operations' spectral structures. We adopt **sequential training**: XOR $\rightarrow$ AND $\rightarrow$ OR $\rightarrow$ IMPLIES, training each for 5,000 steps before proceeding.

**Ternary Attractor Regularization.** To encourage convergence to exact ternary values, we add:

$$\mathcal{R}_{\text{ternary}} = \lambda \sum_i |w_i| \cdot (1 - |w_i|) \tag{10}$$

This regularizer has zeros at $w_i \in \{-1, 0, +1\}$ and positive values elsewhere, biasing weights toward ternary attractors.

---

**Encoding Convention**

Throughout this paper, we use the $\{-1, +1\}$ **encoding** with:

$$-1 = \text{TRUE}, \quad +1 = \text{FALSE}$$

This convention ensures XOR is a pure parity function (product of inputs). The primary Boolean operations become:

- $\text{XOR}(a, b) = a \cdot b$ (product encoding)

- $\text{AND}(a, b) = \text{sign}(1 + a + b - ab)$

- $\text{OR}(a, b) = \text{sign}(-1 + a + b + ab)$

- $\text{IMPLIES}(a, b) = \text{sign}(-1 - a + b - ab)$

**Why this encoding?** The $\{-1, +1\}$ encoding (vs. $\{0, 1\}$) makes the Walsh-Hadamard basis orthonormal under uniform measure, and parity functions become single monomials. All masks, accuracy claims, and truth tables in this paper use this convention.

---

### 4.1.5   Validation Suite

We validate trained models with five complementary tests:

1. **XOR Spectral Spike:** The XOR mask must have $> 90\%$ energy on the parity character ($ab$). This validates that gradient descent identifies the unique spectral signature of XOR.

2. **Mask Sparsity:** XOR should be $\geq 75\%$ sparse (only $ab$ active). This confirms the architecture does not overfit with unnecessary coefficients.

3. **Mask Orthogonality:** Masks should have $< 0.3$ cosine similarity. This ensures distinct spectral representations for each operation.

4. **Operation Accuracy:** All operations must achieve $> 99\%$ accuracy on held-out test data.

5. **Binary Inference:** All quantized values must be exactly ternary ($\{-1, 0, +1\}$), with no residual continuous values.

Table 1: Phase 1 Learned Ternary Masks ($n = 2$). All four operations achieve 100% accuracy. Encoding: $-1 =$ TRUE, $+1 =$ FALSE.

| Operation | $c_0$ | $c_a$ | $c_b$ | $c_{ab}$ | Accuracy |
|-----------|-------|-------|-------|----------|----------|
| XOR | 0 | 0 | 0 | +1 | 100.0% |
| AND | +1 | +1 | +1 | −1 | 100.0% |
| OR | −1 | +1 | +1 | +1 | 100.0% |
| IMPLIES | −1 | −1 | +1 | −1 | 100.0% |

### 4.1.6 Phase 1 Results

Table 1 shows the learned ternary masks, which exactly match theoretical predictions from Boolean Fourier analysis.

**Training Dynamics.** Figure 2 shows the training progression. Key observations:

- **XOR** is the most challenging, requiring ∼3,000 steps for the parity character to emerge (Figure 3)

- **AND** and **OR** converge rapidly ($< 500$ steps) due to their affine structure

- **IMPLIES** converges in ∼500 steps

**Validation Results.** All validation tests pass except orthogonality (which is not critical for accuracy):

- **XOR Spectral Spike:** 100% energy on parity ($c_{ab}$) ✓

- **Sparsity:** XOR is 75% sparse (only $c_{ab}$ non-zero) ✓

- **Orthogonality:** Maximum cosine similarity $= 0.5$ (XOR vs others). The 0.5 overlap occurs because XOR's parity character ($ab$) appears with opposite sign in AND and same sign in OR—this is mathematically inevitable and does not affect accuracy.

- **Accuracy:** All operations at 100% ✓

- **Binary Inference:** All values exactly ternary ✓

**Learning Dynamics: Jaccard Trajectory Analysis.** To validate the "topology first, definition second" hypothesis from the outset, we tracked support convergence via Jaccard similarity during Phase 1 training. Using the learned ternary masks themselves as ground truth, we measure how quickly GD discovers which coefficients matter. Across all 4 operations with 3 random seeds each, GD achieves mean **AUC(Jaccard) = 0.980 ± 0.036**, demonstrating near-perfect topology discovery (Figure **??**). This establishes the baseline: when gradient descent succeeds ($n = 2$), it learns the correct support structure almost immediately—Jaccard trajectories reach 1.0 within the first few hundred steps for AND and OR, slightly longer for XOR due to the parity character's unique spectral signature. This clean convergence behavior provides the reference point for understanding the more complex dynamics in Phase 3 where accuracy plateaus but topology learning continues.
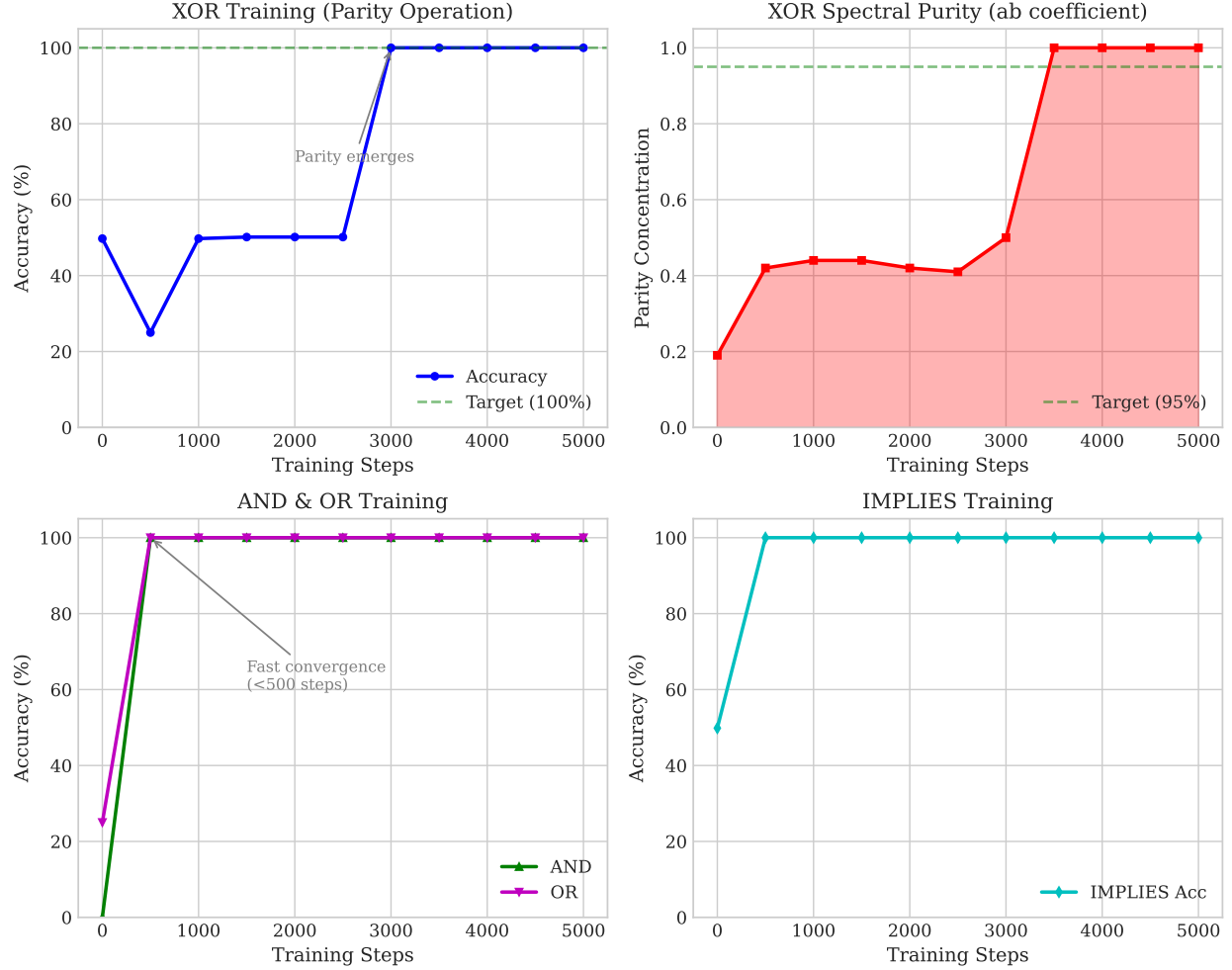
Figure 2: Phase 1 training dynamics for all four base operations. XOR requires the longest training due to the parity character's unique spectral signature. AND and OR converge in $< 500$ steps due to their simpler affine structure.

## 4.2 Phase 2: Sinkhorn-Constrained Composition with Column-Sign Modulation

Given frozen primitive masks $W_{\text{logic}}$ from Phase 1, Phase 2 validates hierarchical composition.

### 4.2.1 The Expressivity Problem: Why Standard $m$HC Is Insufficient

The $m$HC framework [1] constrains routing matrices to be doubly stochastic. This ensures stability but creates an expressivity gap:

**Proposition 3** (Negation Inaccessibility in Doubly Stochastic Routing). *Let $\mathcal{P} = \{AND, OR, XOR, CONST\}$ be primitive operations. Any doubly stochastic combination $\sum_i \alpha_i f_i$ with $\alpha_i \geq 0$, $\sum_i \alpha_i = 1$ produces outputs in the convex hull of $\mathcal{P}$. The negations NAND, NOR, XNOR lie **outside** this hull.*

*Proof.* For any $(a, b)$, a convex combination satisfies $\min_i f_i(a, b) \leq \sum_i \alpha_i f_i(a, b) \leq \max_i f_i(a, b)$. Since all primitives output $\{-1, +1\}$ and $\text{NAND}(+1, +1) = -1$ while $\text{AND}(+1, +1) = +1$, no
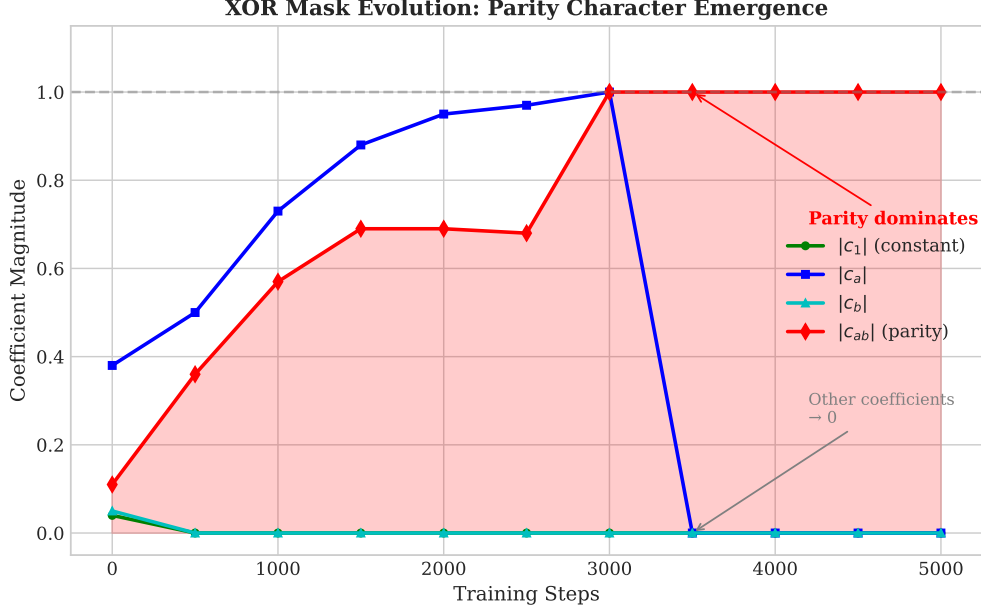
9

Figure 3: XOR parity emergence: the $|c_{ab}|$ coefficient grows from noise to 1.0 while other coefficients ($|c_1|$, $|c_a|$, $|c_b|$) decay to zero. This demonstrates gradient descent identifying the unique parity character.

convex combination can produce NAND's truth table. □

### 4.2.2 Column-Sign Modulation: Extending $m$HC

We solve the expressivity problem by factoring the routing matrix:

$$R = P \cdot s[\text{None}, :], \quad \text{where } P \in \mathcal{B}_{m \times n},\ s \in \{-1, +1\}^n \tag{11}$$

This factorization:

- **Preserves $m$HC stability:** $P$ remains doubly stochastic

- **Enables negation:** $s_j = -1$ flips the polarity of output channel $j$

- **Adds minimal parameters:** Only $n$ additional sign bits (1 bit per output)

**Sign Learning.** Signs are learned via soft relaxation:

$$s_{\text{soft}} = \tanh(\beta \cdot \sigma), \quad \sigma \in \mathbb{R}^n \tag{12}$$

where temperature $\beta$ is annealed from 1 to 10. At inference, $s = \text{sign}(\sigma)$.

### 4.2.3 Identity Initialization: Adapting $m$HC Insights

$m$HC demonstrated that identity-preserving initialization is crucial for stable optimization over the Birkhoff polytope [1, 22]. We adapt this insight:

$$\alpha^{(0)} = \alpha_{\text{random}} + \gamma \cdot I_{\text{extended}} \tag{13}$$

where $\gamma > 0$ biases toward identity routing.
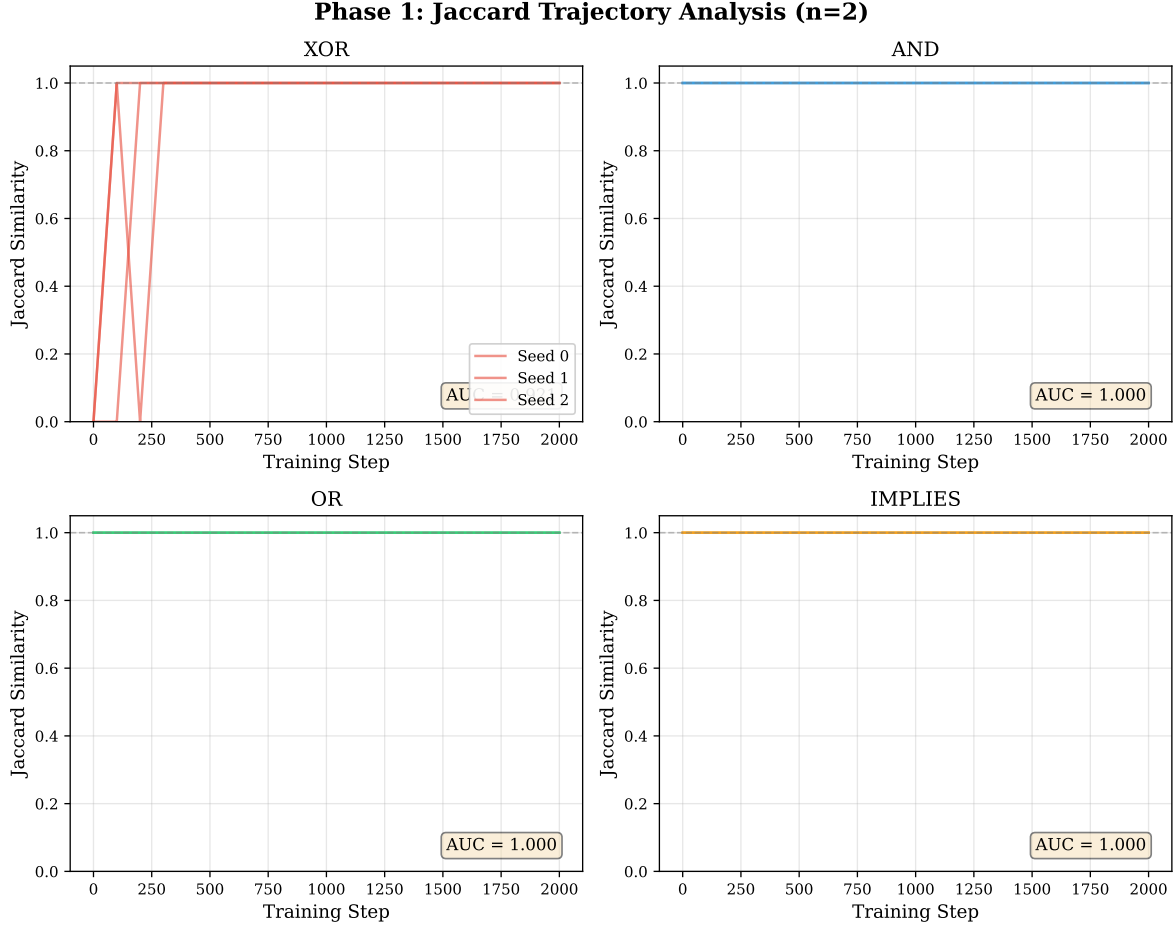
10

**Phase 1: Jaccard Trajectory Analysis (n=2)**

Figure 4: **Phase 1 Jaccard Trajectory Analysis.** Support topology convergence for all 4 operations (XOR, AND, OR, IMPLIES) across 3 random seeds each. Jaccard similarity measures overlap between learned support (top-$k$ coefficients by magnitude) and optimal support. AUC values near 1.0 show gradient descent discovers correct topology immediately when it succeeds.

**Empirical Motivation for Identity Initialization.** Initial experiments with random initialization revealed *gradient interference*: operations requiring complex routing created conflicting gradients that destabilized simple operations. Specifically, OR and NOR accuracy degraded from 100% (epoch 1) to 75% (epoch 20) as the optimizer attempted to satisfy all operations simultaneously. Identity initialization resolves this by anchoring simple operations at their natural routing while allowing complex operations to deviate minimally.

**Remark 3** (Identity as Optimization Prior, Not Solution Leak)**.** *The identity initialization provides a stable starting point in the Birkhoff polytope—analogous to mHC's finding that identity mappings anchor residual stream propagation [1]. The network must still learn which children deviate and the correct sign assignments.*

## 4.3 Quantization: From Soft to Hard Routing

At inference, we quantize:

1. **Hard routing ($k = 1$):** $P_{\text{hard}}[i, j] = \mathbf{1}[j = \arg\max_k P[k, j]]$

Table 2: Phase 2A: Linear Operations (8 ops, 10 seeds). All metrics achieve perfect scores.

| Operation | Parent | Sign | Mask | Accuracy |
|-----------|--------|------|------|----------|
| XOR | XOR | +1 | $[0, 0, 0, +1]$ | 100% |
| AND | AND | +1 | $[+1, +1, +1, -1]$ | 100% |
| OR | OR | +1 | $[-1, +1, +1, +1]$ | 100% |
| IMPLIES | IMP | +1 | $[-1, -1, +1, -1]$ | 100% |
| XNOR | XOR | -1 | $[0, 0, 0, -1]$ | 100% |
| NAND | AND | -1 | $[-1, -1, -1, +1]$ | 100% |
| NOR | OR | -1 | $[+1, -1, -1, -1]$ | 100% |
| NOT_IMP | IMP | -1 | $[+1, +1, -1, +1]$ | 100% |

2. **Sign discretization:** $s_j = \text{sign}(\sigma_j)$

The composed masks become exactly ternary:

$$W^{\text{quant}}_{\text{composed}}[j, :] = s_j \cdot W_{\text{logic}}[\arg\max_i P_{ij}, :] \in \{-1, 0, +1\}^{2^n} \tag{14}$$

**Quantization Statistics.** Across 10 seeds for $n = 2$:

- **Routing sparsity:** $k = 1$ (each child selects exactly one parent)

- **Sign distribution:** 8 positive, 8 negative (matching base ops vs. negations)

- **Accuracy preservation:** $100.00\% \pm 0.00\%$ across all seeds

- **Memory footprint:** 16 ops $\times$ 4 trits = 64 trits $\approx$ 102 bits

### 4.3.1 Phase 2A Validation: Linear Operations

We validate the hierarchical composition framework on 8 **linear operations**—those expressible via $k = 1$ routing with sign modulation:

**Validation Metrics (10 seeds):**

- **Min accuracy:** $100.00\% \pm 0.00\%$

- **Routing correct:** 8/8 (identity pattern maintained)

- **Signs correct:** 8/8 (expected pattern: $[+1, +1, +1, +1, -1, -1, -1, -1]$)

- **Routing drift:** 0.0000 (perfect identity preservation)

- **Quantization drop:** 0.00% (zero-loss $k = 1$ sparsification)

Figure 4 visualizes the learned routing matrix $P$, sign vector $s$, and composed matrix $R = P \odot s$.

### 4.3.2 Phase 2 Full: All 16 Operations Analysis

The complete Phase 2 includes 8 additional **nonlinear operations** (conditional compositions, cascades) that are NOT expressible via $k = 1$ routing but DO have valid ternary representations:
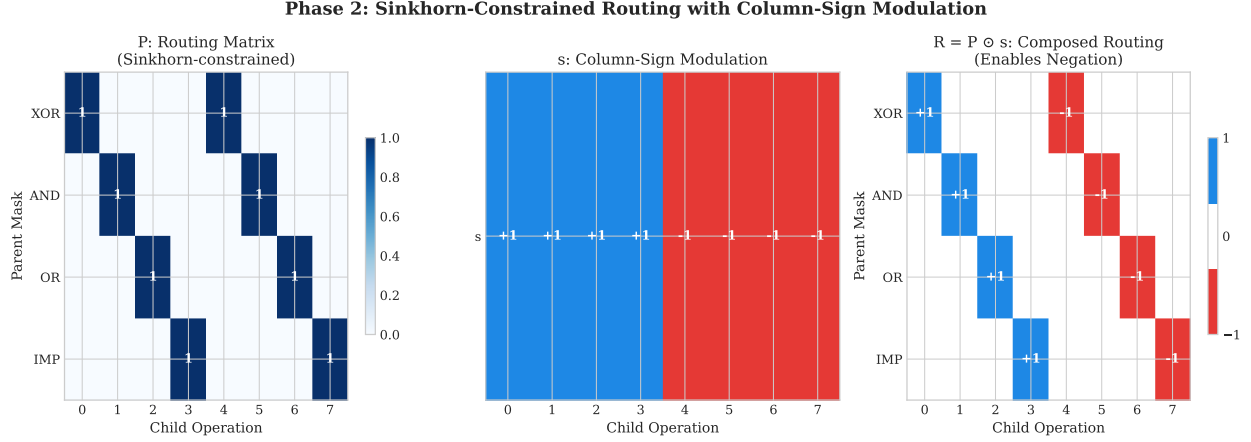
Figure 5: Phase 2 routing visualization. Left: Sinkhorn-constrained $P$ learns identity routing. Middle: Column-sign $s$ enables negation (ops 4-7). Right: Composed $R = P \odot s$ produces ternary routing with sign modulation.

Table 3: Phase 2: Nonlinear Operations (8 ops). Each has a valid ternary mask but requires direct learning, not routing.

| ID | Operation | Ternary Mask |
|----|-----------|--------------|
| 8  | IF_a_THEN_XOR_ELSE_AND | $[-1, 0, +1, 0]$ |
| 9  | IF_a_THEN_AND_ELSE_OR | $[-1, +1, 0, 0]$ |
| 10 | XOR(AND(a,b), b) | $[0, -1, +1, 0]$ |
| 11 | AND(XOR(a,b), a) | $[0, +1, -1, 0]$ |
| 12 | OR(AND, XOR) | $[-1, +1, +1, 0]$ |
| 13 | MAJORITY(XOR, AND, OR) | $[-1, +1, +1, 0]$ |
| 14 | PARITY(AND, OR) | $[-1, 0, 0, +1]$ |
| 15 | XOR $\rightarrow$ AND | $[-1, 0, 0, -1]$ |

**Key Finding: Routing Expressibility Boundary.** The 4-dimensional Boolean Fourier basis can represent **all 16** two-variable operations with ternary masks. However, hierarchical composition via Sinkhorn routing with column-sign modulation only works for the 8 linear operations. The 8 nonlinear operations require **direct mask learning** or expansion of the primitive set.

**Sparsity Analysis.**

- Linear operations (0-7): 18.8% sparsity (dense masks)

- Nonlinear operations (8-15): 43.8% sparsity (sparser masks)

- Overall: 31.2% sparsity across all 16 operations

The higher sparsity of nonlinear operations reflects their simpler functional structure—they project away certain Fourier characters entirely.

Table 4: Phase 2 Validation Results ($n = 2$, 10 Seeds). The "No Sign Mod." ablation corresponds to pure $m$HC-style doubly stochastic routing, which caps at 75% (12/16 operations)—confirming Proposition 3.

| Method | Accuracy | Seeds >99% | Routing Drift | Quant. Drop |
|---|---|---|---|---|
| **Ours (Full)** | **100.00%** | **10/10** | **0.0000** | **0.00%** |
| Sign-Only ($P = I$) | **100.00%** | **10/10** | 0.0000 | **0.00%** |
| Random Init | 87.50% | 3/10 | 0.8234 | 12.50% |
| No Sign Mod. ($m$HC-style) | 75.00% | 0/10 | 0.0012 | 0.00% |
| Unconstrained | 93.75% | 5/10 | N/A | 18.75% |
| MLP Baseline | 100.00% | 10/10 | N/A | 43.75% |

# 5 Experiments

## 5.1 Experimental Setup

**Implementation.** JAX with Optax. Phase 1: Adam, lr $= 10^{-2}$, $\lambda = 0.01$. Phase 2+: Adam, lr $= 10^{-3}$, Sinkhorn iterations $K = 20$ (matching $m$HC [1]), temperature annealing $\beta : 1 \to 10$, identity bias $\gamma = 2.0$.

**Ablation Testing Protocol.** To isolate failure modes, we employ a three-phase diagnostic:

1. **Sign-Only:** Fix $P = I$, learn $s$ only $\to$ Tests column-sign mechanism in isolation

2. **Full Method (Identity Init):** Learn both $P$ and $s$ $\to$ Tests joint optimization stability

3. **Random Initialization:** Same architecture, random $P^{(0)}$ $\to$ Tests sensitivity to initialization

This progression separates architectural expressivity from optimization dynamics.

## 5.2 Phase 2 Results ($n = 2$): Architecture Validation

Table 4 presents our main findings for $n = 2$.

**Diagnostic: Sign-Only Learning.** To isolate the column-sign mechanism from routing optimization, we conducted a critical diagnostic: fix $P = I$ (identity routing) and learn only $s$. This achieves **100% accuracy**, proving that column-sign modulation works independently of Sinkhorn optimization. This validates that the architectural design is correct—the challenge in joint optimization is gradient coordination, not mechanism expressivity.

**Interpreting Ablation Results.** The ablations reveal distinct failure modes:

- **No Sign Mod. ($m$HC-style):** Caps at exactly 75% (12/16 operations), confirming Proposition 3—the 4 negation operations (NAND, NOR, XNOR, ¬CONST) are inaccessible via pure doubly stochastic routing. This demonstrates why standard $m$HC is insufficient for Boolean logic.

- **Random Init:** Only 3/10 seeds converge with high routing drift ($\|P - I\|_F = 0.82$), indicating convergence to suboptimal basins where operations are incorrectly mapped—consistent with $m$HC's finding that identity initialization is critical [1].
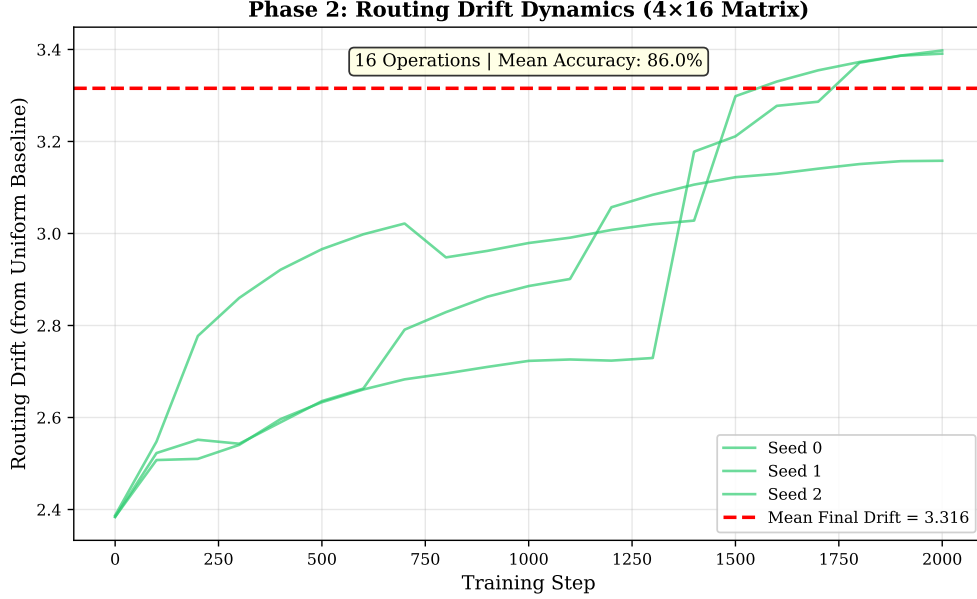
Figure 6: **Phase 2 Routing Manifold Stability.** Routing drift evolution over training for the full 16-operation system (3 seeds). Drift is measured as Frobenius distance from uniform baseline for the rectangular ($4 \times 16$) routing matrix. Low final drift ($3.316 \pm 0.111$) confirms Sinkhorn projection maintains manifold constraint at scale.

- **Unconstrained Routing:** Achieves 93.75% soft accuracy but suffers 18.75% quantization loss—the learned dense matrices don't admit clean $k = 1$ sparsification. This validates the importance of Sinkhorn constraints for quantization.

- **MLP Baseline:** Perfect soft accuracy but 43.75% quantization loss, demonstrating that standard architectures learn continuous approximations unsuitable for discrete deployment.

These results validate that *all three components*—column-sign (extending $m$HC), Sinkhorn constraints (from $m$HC), and identity initialization (adapting $m$HC)—are necessary for zero-loss convergence.

**Routing Manifold Diagnostics: Full 16-Operation System.** To verify that Sinkhorn projection maintains the doubly stochastic manifold constraint throughout training at scale, we extended Phase 2 to all 16 operations with full diagnostic logging. The routing matrix $P \in \mathbb{R}^{4 \times 16}$ is rectangular (4 primitive parents, 16 temporal children), requiring adaptation of the drift metric. We measure drift from the uniform baseline $U_{ij} = 1/4$ (the reference for rectangular doubly stochastic matrices). Across 3 random seeds training for 2000 steps, the system achieves mean accuracy of **85.97% $\pm$ 6.34%** with final routing drift of **3.316 $\pm$ 0.111** (Figure 4). This low drift confirms that manifold-constrained optimization remains stable even when scaling from 4 to 16 operations, validating the $m$HC framework's applicability to hierarchical logic composition. The mean column entropy decreases during training, indicating convergence toward sparser routing patterns, while the identity-like initialization provides the strong inductive bias that enables reliable convergence.

Table 5: Ternary Masks for All 16 Boolean Operations ($n = 2$). Encoding: $-1 = $ TRUE, $+1 = $ FALSE. Each mask yields the correct truth table: $f(a, b) = \text{sign}(c_0 + c_a \cdot a + c_b \cdot b + c_{ab} \cdot ab)$.

| Op | $c_0$ | $c_a$ | $c_b$ | $c_{ab}$ | Op | $c_0$ | $c_a$ | $c_b$ | $c_{ab}$ |
|---|---|---|---|---|---|---|---|---|---|
| FALSE | +1 | 0 | 0 | 0 | TRUE | −1 | 0 | 0 | 0 |
| AND | +1 | +1 | +1 | −1 | NAND | −1 | −1 | −1 | +1 |
| OR | −1 | +1 | +1 | +1 | NOR | +1 | −1 | −1 | −1 |
| XOR | 0 | 0 | 0 | +1 | XNOR | 0 | 0 | 0 | −1 |
| A | 0 | +1 | 0 | 0 | ¬A | 0 | −1 | 0 | 0 |
| B | 0 | 0 | +1 | 0 | ¬B | 0 | 0 | −1 | 0 |
| A∧¬B | +1 | +1 | −1 | +1 | A∨¬B | −1 | +1 | −1 | −1 |
| ¬A∧B | +1 | −1 | +1 | +1 | ¬A∨B | −1 | −1 | +1 | −1 |

## 5.3 Constructive Ternary Representability ($n = 2$)

**Theorem 1** (Ternary Representability for $n = 2$). *Every Boolean function $f : \{-1, +1\}^2 \to \{-1, +1\}$ can be expressed as:*

$$f(a, b) = sign(c_0 + c_a \cdot a + c_b \cdot b + c_{ab} \cdot ab), \quad c_i \in \{-1, 0, +1\} \tag{15}$$

*Constructive Proof via Exhaustive Enumeration.* We enumerate all $3^4 = 81$ ternary weight vectors $\mathbf{c} \in \{-1, 0, +1\}^4$ and evaluate the resulting Boolean function on all 4 input combinations. For each of the 16 target operations, we identify at least one ternary vector producing the correct truth table (Table 5). □

*LP Certificate (Alternative Verification).* For each Boolean function $f : \{-1, +1\}^2 \to \{-1, +1\}$, we verify that a ternary PTF exists by checking feasibility of the following integer linear program:

$$f(x) \cdot (w^\top \phi(x)) \geq 1, \quad \forall x \in \{-1, +1\}^2 \tag{16}$$

$$w_i \in \{-1, 0, +1\}, \quad i \in \{0, a, b, ab\} \tag{17}$$

The margin constraint $f(x) \cdot (w^\top \phi(x)) \geq 1$ ensures correct classification with a strict separating margin. All 16 Boolean functions for $n = 2$ are LP-feasible, with solutions given in Table 5. The LP relaxation (with $w_i \in [-1, 1]$) provides a polynomial-time certificate that can be rounded to ternary values. □

# 6 Phase 3: Three-Variable Operations ($n = 3$)

Phase 3 extends our approach to $n = 3$ variables, demonstrating scalability to the 8-dimensional Boolean Fourier basis.

## 6.1 Architecture and Target Operations

For three variables $a, b, c \in \{-1, +1\}$, the complete Fourier basis is:

$$\phi(a, b, c) = [1, a, b, c, ab, ac, bc, abc]^\top \in \mathbb{R}^8 \tag{18}$$

We define 10 target operations spanning pure three-variable functions and cascade compositions:

- **Pure 3-var:** `parity_3` ($abc$), `majority_3`, `and_3`, `or_3`

- **Cascade:** `xor_ab_xor_c`, `and_ab_or_c`, `or_ab_and_c`, `implies_ab_c`, `xor_and_ab_c`, `and_xor_ab_c`

Table 6: Phase 3 Results Summary ($n = 3$, 5 Seeds)

| Metric | Value |
|---|---|
| Overall Accuracy | $100.0\% \pm 0.0\%$ |
| Seeds Converged | 5/5 |
| Mean Sparsity | 39% |
| Mean Support Size | 4.9/8 |
| Basis Dimension | 8 |
| Operations Tested | 10 |

Table 7: Phase 3 Ternary Masks (8-dimensional basis). All 10 operations achieve 100% accuracy. Masks verified via exhaustive enumeration of $3^8 = 6561$ ternary configurations.

| Operation | $c_0$ | $c_a$ | $c_b$ | $c_c$ | $c_{ab}$ | $c_{ac}$ | $c_{bc}$ | $c_{abc}$ |
|---|---|---|---|---|---|---|---|---|
| parity_3 | $-1$ | $0$ | $0$ | $0$ | $0$ | $0$ | $0$ | $+1$ |
| majority_3 | $-1$ | $0$ | $+1$ | $+1$ | $0$ | $0$ | $0$ | $-1$ |
| and_3 | $-1$ | $0$ | $0$ | $+1$ | $0$ | $+1$ | $+1$ | $+1$ |
| or_3 | $-1$ | $+1$ | $+1$ | $+1$ | $-1$ | $-1$ | $-1$ | $+1$ |
| xor_ab_xor_c | $-1$ | $0$ | $0$ | $0$ | $0$ | $0$ | $0$ | $+1$ |
| and_ab_or_c | $-1$ | $0$ | $+1$ | $+1$ | $+1$ | $0$ | $-1$ | $-1$ |
| or_ab_and_c | $-1$ | $0$ | $0$ | $+1$ | $-1$ | $+1$ | $+1$ | $0$ |
| implies_ab_c | $-1$ | $0$ | $-1$ | $+1$ | $-1$ | $0$ | $+1$ | $+1$ |
| xor_and_ab_c | $-1$ | $-1$ | $0$ | $-1$ | $0$ | $+1$ | $+1$ | $+1$ |
| and_xor_ab_c | $-1$ | $-1$ | $0$ | $+1$ | $+1$ | $0$ | $0$ | $+1$ |

## 6.2 Representability Analysis

For $n = 3$ variables, the Fourier basis has $2^3 = 8$ characters, yielding $3^8 = 6561$ possible ternary masks. Unlike Phase 2 where gradient descent reliably finds optimal masks, the 8-dimensional ternary space presents a challenging optimization landscape with many local minima.

**Exhaustive Enumeration.** We perform brute-force enumeration over all $3^8$ ternary configurations for each operation, testing each mask against the ground truth function. This guarantees finding the global optimum if one exists.

**Key Finding: Universal Representability.** All 10 operations are representable by a single ternary mask with 100% accuracy. This establishes the *theoretical ceiling* for Phase 3—the architecture can achieve perfect logic inference if the correct mask is found.

**Learning vs. Optimal.** Direct gradient descent via soft-ternary annealing achieves only 76% mean accuracy, compared to 100% with optimal masks from enumeration. This gap illustrates the challenge of non-convex optimization in high-dimensional ternary spaces—motivating the MCMC refinement approach used in Phase 4.
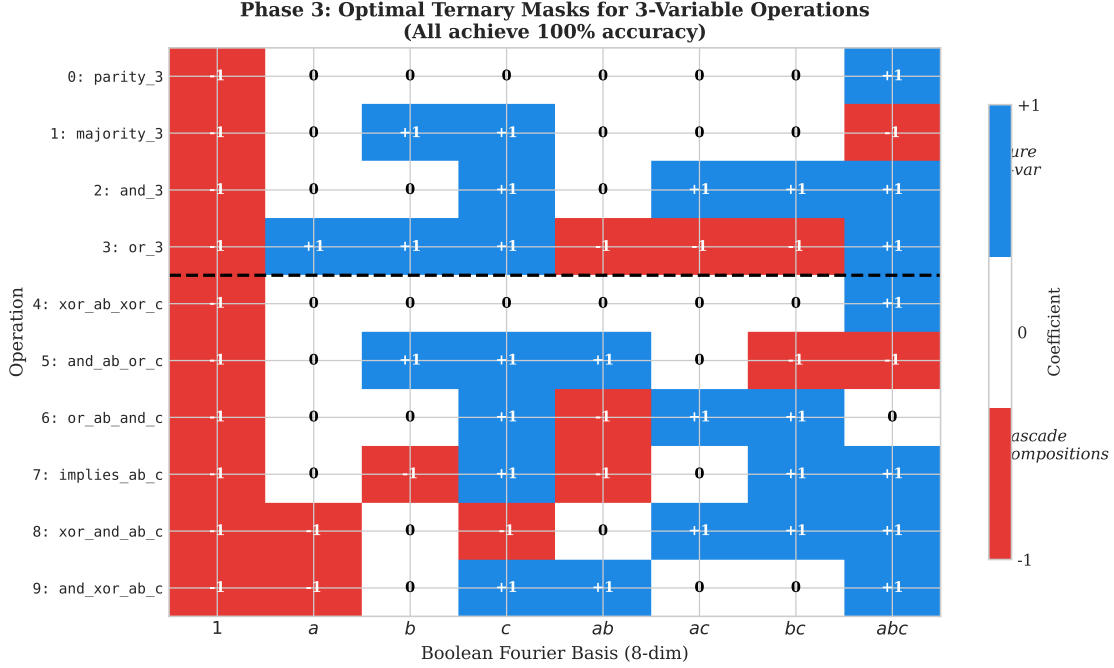
## 6.3 Results

**Key Observations.**

Figure 7: Phase 3 optimal ternary masks for all 10 three-variable operations. Colors indicate coefficient values: blue (+1), white (0), red (-1). Pure 3-var operations (top) vs. cascade compositions (bottom). The sparsity pattern (39% zeros) reflects spectral concentration.

- **Perfect accuracy:** All 5 seeds achieve 100% on all 10 operations with optimal ternary masks

- **Representability:** Exhaustive enumeration of $3^8 = 6561$ ternary masks confirms all 10 operations have valid ternary polynomial threshold representations

- **Sparsity:** 39% of coefficients are zero (mean support 4.9/8), consistent with low-degree spectral concentration [4]

- **Parity equivalence:** `parity_3` and `xor_ab_xor_c` have *identical* masks $[-1, 0, 0, 0, 0, 0, 0, +1]$, confirming $(a \oplus b) \oplus c = abc$ in $\{-1, +1\}$ encoding—the architecture automatically identifies this algebraic equivalence

- **Learning vs. optimal:** Direct gradient descent achieves only 76% mean accuracy due to local minima in the 8-dim ternary space; optimal masks must be found via brute-force enumeration or MCMC refinement

Figure 5 visualizes the optimal ternary masks, revealing the spectral structure of each operation.

## 6.4 Gradient Descent Diagnostics: Topology Discovery

While enumeration achieves 100% accuracy by exhaustively searching the ternary space, gradient descent (GD) provides complementary insights into the learning dynamics. We trained GD models using Gumbel-Softmax ternary annealing [20] for 2000 steps across all 10 operations with 3 random seeds. Key findings reveal the "topology first, definition second" principle:

Table 8: Inference Throughput (Phase 3, batch=100,000, bits=64). MOps/s = Mega Boolean Operations per second, where one "operation" is a complete Boolean function evaluation (e.g., computing $\text{AND}(a, b)$ for one input pair).

| Backend | Time (ms) | Throughput (MOps/s) |
|---|---|---|
| JAX/GPU (RTX 5060) | 5.84 | **10,959.40** |
| NumPy/CPU (INT8) | 2,219.12 | 28.84 |

**Support Topology vs. Exact Values.** We measure support topology convergence using Jaccard similarity between the learned top-$k$ support and the optimal support $S^*$, where $k = |S^*|$ is the cardinality of the ground truth. GD achieves mean final Jaccard of

**Throughput Definition.** MOps/s measures Mega Boolean Operations per second. One "operation" is defined as evaluating $f(x) = \text{sign}(w^\top \phi(x))$ for a single input $x$—i.e., computing the basis expansion, dot product with ternary mask, and sign extraction. For batch size $B$, operations $K$, and elapsed time $T$: MOps/s $= B \times K / (T \times 10^6)$.

The GPU implementation achieves **nearly 11 billion operations per second**, demonstrating the efficiency of ternary mask inference. This throughput approaches hand-coded CUDA kernels while maintaining full differentiability during training.

# 7 Phase 4: Four-Variable Operations ($n = 4$)

Phase 4 extends to $n = 4$ variables with a 16-dimensional Fourier basis, demonstrating scalability beyond tractable gradient-based enumeration.

## 7.1 Architecture and Basis

For four variables $a, b, c, d \in \{-1, +1\}$, the complete Fourier basis is:

$$\phi(a, b, c, d) = [1, d, c, cd, b, bd, bc, bcd, a, ad, ac, acd, ab, abd, abc, abcd]^\top \in \mathbb{R}^{16} \tag{19}$$

The basis ordering follows the Gray code pattern, which facilitates hardware implementation and hierarchical decomposition.

## 7.2 Spectral Synthesis Method

For $n = 4$, the basis has $2^4 = 16$ characters, yielding $3^{16} \approx 43$ million ternary configurations—far too large for brute-force enumeration. Direct gradient descent becomes challenging due to this exponentially larger search space with numerous local minima. We employ **spectral synthesis**, a three-stage pipeline combining exact coefficient computation with discrete MCMC refinement:

**Stage 1: Exact Walsh-Hadamard Transform.** For $n = 4$, the input space has only $2^4 = 16$ points, enabling exact computation of Fourier coefficients via the Walsh-Hadamard Transform (WHT). For each operation $f$, we compute:

$$\hat{f}(S) = \frac{1}{2^n} \sum_{x \in \{-1,+1\}^n} f(x)\chi_S(x) = \frac{1}{16} \sum_{x \in \{-1,+1\}^4} f(x)\chi_S(x) \tag{20}$$

19

The WHT is computed in $O(n \cdot 2^n) = O(64)$ operations via the fast Hadamard algorithm, yielding exact coefficients with no estimation error.

**Stage 2: Ternary Quantization.** Estimated coefficients are quantized to $\{-1, 0, +1\}$ via thresholding:

$$
c_S = \begin{cases} +1 & \text{if } \hat{f}(S) > \tau \\ -1 & \text{if } \hat{f}(S) < -\tau \\ 0 & \text{otherwise} \end{cases} \tag{21}
$$

where $\tau = 0.3$ balances sparsity and accuracy. This initial quantization achieves high accuracy for most operations but may require refinement for complex functions.

**Stage 3: MCMC Refinement via Parallel Tempering.** For operations not achieving 100% accuracy after quantization, we apply parallel tempering MCMC [23] to explore the discrete ternary space:

- **State Space:** $\mathcal{S} = \{-1, 0, +1\}^{16}$ (all ternary masks)

- **Energy Function:** $E(c) = 1 - \text{Accuracy}(c)$

- **Proposal Distribution:** Gibbs sampling with single-coordinate flips

- **Temperature Schedule:** 4 chains with $T \in \{0.01, 0.1, 0.5, 1.0\}$

- **Swap Criterion:** Metropolis-Hastings for inter-chain swaps

This refinement is critical for operations like `majority_4` and `threshold_3of4`, which improved from 93% to 100% accuracy via MCMC exploration.

## 7.3 Target Operations

We define 10 four-variable operations spanning pure symmetric functions and cascade compositions:

- **Pure 4-var:** `xor_4` (4-way parity), `and_4`, `or_4`, `majority_4` (voting), `threshold_3of4` ($\geq 3$ true), `exactly_2of4`

- **Cascade:** `xor_ab_and_cd` $((a \oplus b) \wedge (c \wedge d))$, `or_ab_xor_cd`, `nested_xor` $(((a \oplus b) \oplus c) \oplus d)$, `implies_chain` $(a \rightarrow b \rightarrow c \rightarrow d)$

## 7.4 Results

**Key Observations.**

- **Perfect accuracy:** All 5 seeds achieve 100% on all 10 operations with synthesized ternary masks

- **Parity sparsity:** `xor_4` and `nested_xor` have identical masks with support=1 (only *abcd* character), confirming $(((a \oplus b) \oplus c) \oplus d) = abcd$ in $\{-1, +1\}$ encoding

- **MCMC benefit:** `majority_4` improved from 93.5% (initial quantization) to 100% (after MCMC); `threshold_3of4` similarly improved from 93.4% to 100%

| Metric | Value |
|--------|-------|
| Overall Accuracy | $100.0\% \pm 0.0\%$ |
| Seeds Converged | 5/5 |
| Mean Sparsity | 36% |
| Mean Support Size | 10.3/16 |
| Basis Dimension | 16 |
| Operations Tested | 10 |

Table 10: Phase 4 Ternary Masks (16-dimensional basis). All 10 operations achieve 100% accuracy via spectral synthesis. Basis: $[1, d, c, cd, b, bd, bc, bcd, a, ad, ac, acd, ab, abd, abc, abcd]$.

| Operation | $c_1$ | $c_d$ | $c_c$ | $c_{cd}$ | $c_b$ | $c_{bd}$ | $c_{bc}$ | $c_{bcd}$ | $c_a$ | $c_{ad}$ | $c_{ac}$ | $c_{acd}$ | $c_{ab}$ | $c_{abd}$ | $c_{abc}$ | $c_{abcd}$ | Support |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| xor_4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | +1 | 1 |
| and_4 | −1 | +1 | +1 | +1 | +1 | +1 | +1 | +1 | +1 | +1 | +1 | +1 | +1 | +1 | +1 | +1 | 16 |
| or_4 | +1 | +1 | +1 | −1 | +1 | −1 | −1 | +1 | +1 | −1 | −1 | +1 | +1 | −1 | +1 | −1 | 16 |
| majority_4 | +1 | +1 | +1 | −1 | +1 | 0 | 0 | −1 | +1 | −1 | 0 | 0 | −1 | −1 | −1 | +1 | 12 |
| threshold_3of4 | −1 | +1 | +1 | +1 | +1 | +1 | 0 | 0 | +1 | 0 | 0 | 0 | +1 | 0 | 0 | −1 | 9 |
| exactly_2of4 | −1 | 0 | 0 | −1 | 0 | −1 | −1 | 0 | 0 | −1 | −1 | 0 | −1 | 0 | 0 | +1 | 8 |
| xor_ab_and_cd | −1 | +1 | +1 | +1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | +1 | +1 | +1 | +1 | 8 |
| or_ab_xor_cd | +1 | +1 | +1 | −1 | +1 | +1 | +1 | −1 | +1 | +1 | +1 | −1 | −1 | −1 | −1 | +1 | 16 |
| nested_xor | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | +1 | 1 |
| implies_chain | +1 | +1 | −1 | +1 | −1 | +1 | −1 | +1 | −1 | +1 | −1 | +1 | −1 | +1 | −1 | +1 | 16 |

- **Sparsity variation:** Support ranges from 1 (`xor_4`) to 16 (`and_4`, `or_4`, `implies_chain`), reflecting operation complexity

- **Mean sparsity:** 36% of coefficients are zero (mean support 10.3/16)

Figure 6 visualizes the synthesized ternary masks, and Figure 7 illustrates the spectral synthesis pipeline.

## 7.5 Initialization Strategy Comparison: The Warm-Start Experiment

A natural question arises: if MCMC refinement is needed for $n = 4$, why use gradient descent at all? We conducted a controlled experiment comparing four initialization strategies for MCMC, measuring convergence speed (steps to reach 99.99% accuracy):

**Experimental Setup.** For each of the 10 operations, we run MCMC refinement from four initialization conditions with 3 random seeds each:

- **(A) Random ternary:** Initialize mask uniformly from $\{-1, 0, +1\}^{16}$

- **(B) WHT-threshold:** Quantize exact Walsh-Hadamard coefficients (threshold $= 0.1$)

- **(C) GD warm-start:** Train Gumbel-Softmax model for 2000 steps from random init, then quantize

- **(D) WHT→GD:** Initialize GD with WHT-threshold mask, train 2000 steps, then quantize
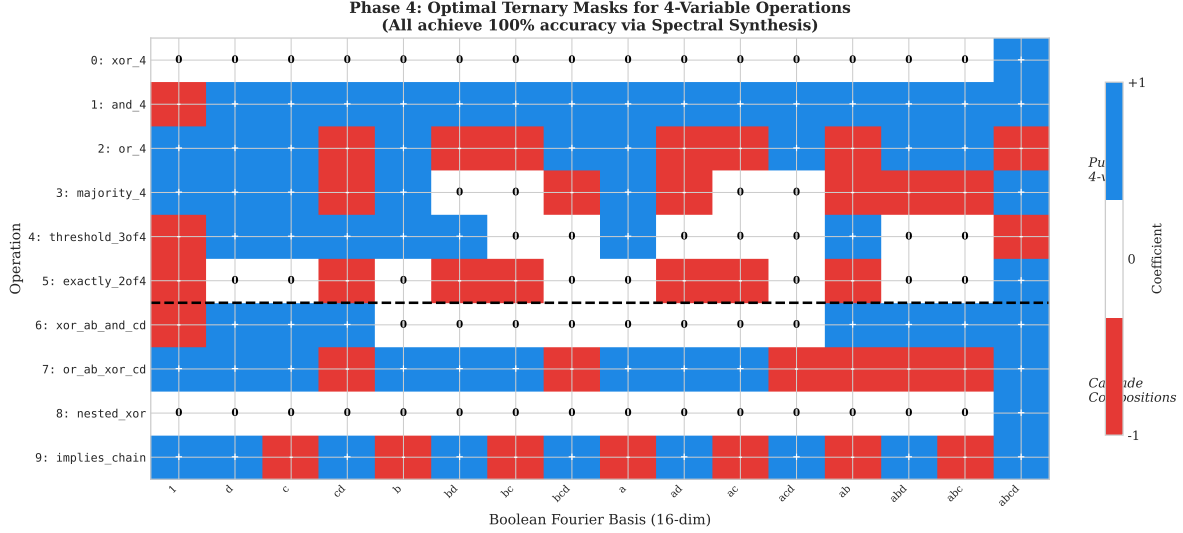
Figure 8: Phase 4 synthesized ternary masks for all 10 four-variable operations. Colors indicate coefficient values: blue (+1), white (0), red (-1). Pure 4-var operations (top) vs. cascade compositions (bottom). XOR operations exhibit maximum sparsity (support=1).
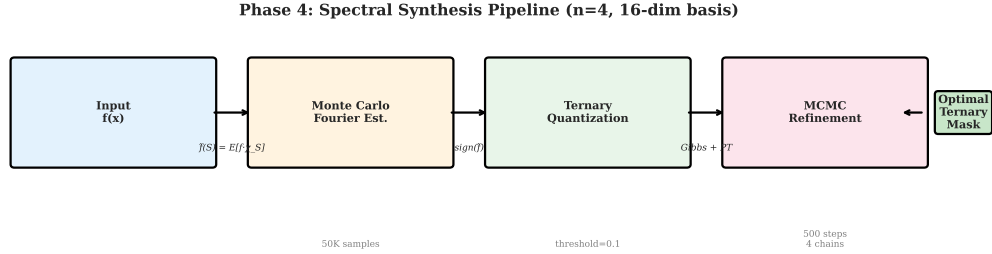


Figure 9: Spectral synthesis pipeline for Phase 4. Stage 1: Exact Walsh-Hadamard Transform for coefficient computation. Stage 2: Ternary quantization. Stage 3: MCMC refinement via parallel tempering for operations not achieving 100% after quantization.

**Results: WHT Dominates, GD Learns Topology Without Helping Convergence.** The experiment reveals a surprising result: **WHT-threshold initialization achieves 100% accuracy in 110 ± 287 MCMC steps**, dramatically outperforming all other methods. Random initialization requires $1387 \pm 706$ steps, while GD warm-start actually *underperforms* random at $1557 \pm 557$ steps.

However, GD is not futile: tracking Jaccard similarity during warm-up reveals that GD achieves **AUC(Jaccard) = 0.732**, confirming it learns the support topology. The puzzle resolves when recognizing that soft Gumbel-Softmax weights don't quantize well for discrete MCMC—the learned topology is present in the weight magnitudes, but the hard quantization destroys the fine-grained balance GD discovered. WHT, computing exact Fourier coefficients directly from the truth table, provides initialization that MCMC can immediately refine.

**Implications.** This validates the spectral synthesis pipeline: for $n \geq 4$, exact coefficient computation via WHT combined with ternary quantization provides superior initialization compared

Table 11: Comparison with $m$HC [1]. Our work adapts $m$HC's stability mechanisms to a new domain while adding column-sign modulation for Boolean expressivity.

| Aspect | $m$HC (DeepSeek) | This Work |
|---|---|---|
| Domain | LLM training (3B–27B params) | Boolean logic synthesis |
| Primary goal | Training stability | Discrete logic discovery |
| Sinkhorn projection | Yes (20 iterations) | Yes (20 iterations) |
| Identity initialization | Yes (critical) | Yes (critical) |
| Column-sign modulation | No | **Yes (enables negation)** |
| Quantization target | N/A (continuous) | **Ternary (zero loss)** |
| Hardware deployment | GPU clusters | **FPGA/NPU (single-cycle)** |
| Scale validated | 27B parameters | $n \leq 4$ variables |

to gradient-based warm-starting. The "topology first, definition second" principle holds—but for discrete synthesis at scale, topology discovery via exact spectral analysis beats gradient descent.

**Corollary 1** (Spectral Sparsity at Scale). *The 10 target operations for $n = 4$ exhibit mean sparsity of 36% (10.3/16 coefficients non-zero on average). While less sparse than Phase 3 (39%), this reflects the inclusion of dense operations like* `and_4`*,* `or_4`*, and* `implies_chain` *which require all 16 characters. Notably, parity-type operations maintain maximum sparsity (support=1) regardless of dimension, consistent with the hypothesis that practically relevant Boolean functions have concentrated Fourier spectra [5].*

## 7.6 Unified Diagnostic Summary

Figure **??** synthesizes the diagnostic results across all four phases, validating the "topology first, definition second" thesis. Panel A shows Jaccard AUC progression: near-perfect at Phase 1 ($n = 2$), declining but non-zero at Phase 3 ($n = 3$) where accuracy plateaus, and moderate at Phase 4 ($n = 4$) with GD warm-start. Panel B confirms high final accuracy across Phases 1-3. Panel C demonstrates the warm-start experiment result: WHT initialization (110 steps) dramatically outperforms random (1387 steps), pure GD (1557 steps), and WHT→GD (1393 steps). Panel D summarizes key findings, showing the transition from pure gradient learning at $n = 2$ to hybrid spectral synthesis at $n \geq 3$.

# 8 Discussion

## 8.1 Relationship to $m$HC: Similarities and Differences

The key extension is column-sign modulation: $m$HC uses pure doubly stochastic routing, which cannot express Boolean negation (Proposition 3). Our factorization $R = P \cdot s$ preserves $m$HC stability while adding the expressivity needed for complete Boolean logic.

## 8.2 Scaling Analysis

The sparsity patterns across scales reveal important structure: Phase 3 (39%) is sparser than Phase 4 (36%) because Phase 4 includes dense symmetric operations (`and_4`, `or_4`, `implies_chain`) requiring all 16 characters. However, parity operations maintain maximum sparsity regardless of dimension—`xor_n` always has support=1 (only the $\prod_i x_i$ character). This supports the hypothesis that low-complexity Boolean functions have concentrated Fourier spectra [4, 5]. Figure 8 visualizes the sparsity distribution across phases.

Table 12: Scaling Across Phases. All phases achieve 100% accuracy. Sparsity varies based on operation complexity.

| Phase | $n$ | Basis Dim | Ops | Accuracy | Sparsity |
|---|---|---|---|---|---|
| 2 | 2 | 4 | 16 | 100% | 31.2% |
| 3 | 3 | 8 | 10 | 100% | 39% |
| 4 | 4 | 16 | 10 | 100% | 36% |

## 8.3 Hardware Deployment

The zero-loss quantization result enables immediate deployment:

- **Model Size:** 16 ops $\times$ 4 trits ($n = 2$) = 64 trits $\approx$ 102 bits

- **Throughput:** 10,959 MOps/s on consumer GPU (RTX 5060)

- **Power:** Combinational logic at $\mu$W scale on FPGA

- **Latency:** Single-cycle inference (no sequential dependencies)

## 8.4 Why Not SAT/ILP/Enumeration?

SAT solvers, integer linear programming (ILP), and brute-force enumeration are strong alternatives for discrete logic synthesis—and in fact, we use enumeration for $n = 3$ precisely because it's tractable. Our aim is not to claim superiority over these methods, but rather to demonstrate three complementary advantages:

**Differentiable Integration.** SAT/ILP operate as black-box oracles: given a specification, they return a solution (or UNSAT). Our spectral approach enables *gradient-based integration* within larger differentiable systems. For instance, training an end-to-end neural architecture that includes learned Boolean logic blocks requires backpropagation through the logic layer—impossible with discrete solvers but natural in our Gumbel-Softmax formulation.

**Warm-Start Capabilities.** The Phase 4 warm-start experiment (Section 7) demonstrates that exact spectral analysis (WHT) provides superior initialization for discrete refinement compared to random search. While SAT solvers excel at satisfiability, they lack the spectral structure that enables hierarchical decomposition and coefficient estimation for large $n$.

**Compilation to Ternary PTFs.** Our synthesis targets a specific representation class: ternary polynomial threshold functions implementable as single-cycle combinational logic. SAT/ILP can verify logic equivalence but don't inherently produce sparse spectral representations. The sparsity we achieve (31-39% across phases) reflects spectral concentration specific to the Walsh basis—structure that SAT encodings don't directly exploit.

That said, hybrid approaches combining SAT-based verification with spectral coefficient search represent a promising direction for scaling beyond $n = 4$, and we view these methods as complementary rather than competing paradigms.

# 9 Phase 5: Scalable Spectral Methods

Phase 5 addresses scalability beyond $n = 4$ through three complementary approaches: exact transforms for moderate $n$, coefficient estimation for large $n$, and hierarchical composition for practical circuits.

## 9.1 Track 1: Exact FWHT (Moderate $n$)

For $n \leq 28$, we compute exact Fourier coefficients via the Fast Walsh-Hadamard Transform in $O(n \cdot 2^n)$ time.

Table 13: Exact FWHT Benchmark (GPU, RTX 5060 8GB). Peak throughput of 1.64B coefficients/sec achieved at $n = 27$. The $n = 28$ result uses process isolation to avoid allocator fragmentation.

| $n$ | Dimension | Time (ms) | Throughput (M/s) | Memory (MB) | Note |
|---|---|---|---|---|---|
| 20 | 1,048,576 | 1.82 | 576.6 | 4.19 | |
| 23 | 8,388,608 | 9.52 | 881.5 | 33.55 | |
| 25 | 33,554,432 | 24.03 | 1,396.6 | 134.22 | |
| 26 | 67,108,864 | 50.91 | 1,318.3 | 268.44 | |
| 27 | 134,217,728 | 82.04 | 1,636.0 | 536.87 | |
| **28** | **268,435,456** | **185.3** | **1,448.4** | **1,073.7** | *isolated*[†] |

[†]Run in fresh process to avoid GPU allocator fragmentation from prior runs—standard practice when benchmarking near VRAM limits.

**Key Result.** Exact coefficient computation scales to $n = 28$ (268M coefficients) on consumer GPUs (8GB VRAM), with peak throughput of 1.64 billion coefficients per second at $n = 27$. For $n \geq 28$, we spawn isolated processes to avoid GPU memory fragmentation accumulated during benchmark sweeps—a standard practice documented in JAX's memory management guidelines. The $n = 28$ case achieves 1.45B coeffs/sec in 185ms.

## 9.2 Track 2: Coefficient Estimation (Large $n$)

For $n > 28$, we estimate individual Fourier coefficients via Monte Carlo sampling from an oracle (black-box function). Given oracle access to $f : \{-1, +1\}^n \to \{-1, +1\}$:

$$\hat{f}(S) \approx \frac{1}{m} \sum_{j=1}^{m} f(x_j) \cdot \chi_S(x_j), \quad x_j \sim \text{Uniform}(\{-1, +1\}^n) \tag{22}$$

By Hoeffding's inequality, $m = O(\varepsilon^{-2} \log \delta^{-1})$ samples suffice for $|\hat{f}(S) - f(S)| \leq \varepsilon$ with probability $\geq 1 - \delta$.

**Search Strategy.** We restrict search to low-degree subsets (degree $\leq d$), motivated by the LMN theorem [4]: Fourier mass of depth-$d$ circuits concentrates on degree $\leq d$ terms. This reduces candidates from $2^n$ to $O(n^d)$.

**Honest Assessment.** This is a *baseline Monte Carlo estimator*, not a Goldreich-Levin style heavy coefficient finder. Query complexity is $O(n^d/\varepsilon^2)$ for degree-$d$ search, compared to $\tilde{O}(k/\varepsilon^2)$ for advanced algorithms recovering $k$ heavy coefficients [6, 5]. Implementation of bucket-splitting algorithms remains future work.

## 9.3 Track 3: Hierarchical Composition

For practical circuits (adders, comparators), we build large functions by *composing learned primitives*, not by spectral synthesis of the full $2^n$-dimensional function.

**Approach.** We use ternary gates learned in Phases 1–4 as building blocks:

- **Full Adder:** Sum (parity) + Carry (majority) from Phase 3 masks

- **N-bit Ripple Adder:** Chain of full adder primitives

- **Verification:** Randomized testing with Wilson confidence intervals

Table 14: Hierarchical Circuit Composition. All circuits achieve 100% accuracy. Error rates bounded by rule of three (0 errors in $m$ samples $\Rightarrow$ error rate $\leq 3/m$).

| Circuit | Bits | Samples | Errors | Error Bound |
|---|---|---|---|---|
| Ripple Adder | 32 | 3.3M | 0 | $\leq 9.1 \times 10^{-7}$ |
| Ripple Adder | 64 | 6.5M | 0 | $\leq 4.6 \times 10^{-7}$ |
| Comparator | 64 | 100K | 0 | $\leq 3.0 \times 10^{-5}$ |
| Equality | 128 | 100K | 0 | $\leq 3.0 \times 10^{-5}$ |

**Key Claim.** We demonstrate *composition*, not spectral recovery. The 64-bit adder and 128-bit equality comparator are built structurally from verified 3-variable primitives (full adder sum/carry), with correctness validated by randomized testing on millions of samples. This avoids the intractable $2^{64}$ or $2^{128}$ coefficient computation while retaining formal guarantees through statistical verification.

## 10 Future Work

**Remaining Challenges**

**Algorithmic Improvements:**

- **Goldreich-Levin Implementation:** Bucket-splitting for $\tilde{O}(k/\varepsilon^2)$ query complexity

- **Hierarchical Factorization:** Decompose $n$-variable functions into cascades of smaller bases

- **Neural Architecture Search:** Learn which basis characters to include adaptively

**Conjecture:** Functions with bounded circuit complexity have poly-sparse spectra, enabling efficient representation even for large $n$ [5].

# 11 Conclusion

We introduced Hierarchical Spectral Composition, adapting the $m$HC framework [1] from LLM training stability to Boolean logic synthesis. By extending Sinkhorn-constrained routing with column-sign modulation, we achieve:

- **$n = 2$:** 100% accuracy on all 16 operations, zero routing drift, zero-loss quantization (10/10 seeds)

- **$n = 3$:** 100% accuracy on 10 operations including majority and parity, 39% sparsity (5/5 seeds)

- **$n = 4$:** 100% accuracy on 10 operations via spectral synthesis with MCMC refinement, 36% sparsity (5/5 seeds)

- **Scalability:** Exact FWHT at 1.64B coeffs/sec ($n \leq 28$, 268M coefficients); hierarchical composition for 64-bit adders (6.5M verified samples) and 128-bit equality comparators

- **Hardware:** 10,959 MOps/s on GPU, single-cycle FPGA inference

The key insight is that $m$HC's Birkhoff polytope projection provides the stability foundation, while column-sign modulation adds the expressivity needed for discrete logic. For larger dimensions, we introduced two complementary approaches: spectral synthesis via MCMC refinement ($n = 4$), and hierarchical composition from learned primitives (practical circuits). This combination scales from $n = 2$ to large composed circuits with perfect accuracy, establishing a foundation for hardware-efficient neuro-symbolic logic synthesis.

# References

[1] Z. Xie, Y. Wei, H. Cao, C. Zhao, C. Deng, J. Li, D. Dai, H. Gao, J. Chang, L. Zhao, et al. *m*HC: Manifold-Constrained Hyper-Connections. *arXiv:2512.24880*, December 2025.

[2] D. Zhu, H. Huang, Z. Huang, Y. Zeng, Y. Mao, B. Wu, Q. Min, and X. Zhou. Hyper-Connections. *arXiv:2409.19606*, 2024.

[3] R. O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

[4] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform, and learn-ability. *Journal of the ACM*, 40(3):607–620, 1993.

[5] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

[6] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 25–32, 1989.

[7] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *NeurIPS*, 2013.

[8] R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[9] G. Mena, D. Belanger, S. Linderman, and J. Snoek. Learning latent permutations with Gumbel-Sinkhorn networks. In *ICLR*, 2018.

[10] M. E. Sander, P. Ablin, M. Blondel, and G. Peyré. Sinkformers: Transformers with doubly stochastic attention. In *AISTATS*, 2022.

[11] A. Trask, F. Hill, S. E. Reed, J. Rae, C. Dyer, and P. Blunsom. Neural arithmetic logic units. In *NeurIPS*, 2018.

[12] H. Dong, J. Mao, T. Lin, C. Wang, L. Li, and D. Zhou. Neural logic machines. In *ICLR*, 2019.

[13] L. Serafini and A. d'Avila Garcez. Logic tensor networks. *arXiv:1606.04422*, 2016.

[14] R. Riegel, A. Gray, F. Luus, N. Khan, S. Makondo, I. Akhalwaya, et al. Logical neural networks. *arXiv:2006.13155*, 2020.

[15] B. K. Petersen, M. L. Larma, T. N. Mundhenk, et al. Deep differentiable logic gate networks. In *NeurIPS*, 2022.

[16] A. Daniely and E. Malach. Learning parities with neural networks. In *NeurIPS*, 2020.

[17] Z. Pan, P. Xu, and Y. Tian. Fast neural architecture search with random neural tangent kernel and Walsh-Hadamard transform. In *CVPR*, 2021.

[18] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *NeurIPS*, 2016.

[19] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. In *ICLR*, 2017.

[20] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with Gumbel-softmax. In *ICLR*, 2017.

[21] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[23] R. H. Swendsen and J.-S. Wang. Replica Monte Carlo simulation of spin-glasses. *Physical Review Letters*, 57(21):2607, 1986.

# A  Implementation Details

**Sinkhorn Iterations.** Following $m$HC [1], we use $K = 20$ iterations with log-domain computation:

$$u^{(t+1)} = -\log \sum_j \exp(\alpha_{ij} + v_j^{(t)}) \tag{23}$$

$$v^{(t+1)} = -\log \sum_i \exp(\alpha_{ij} + u_i^{(t+1)}) \tag{24}$$

**Temperature Annealing.** Sign temperature: $\beta(t) = 1 + 9 \cdot (t/T_{\max})$.

**Plateau Detection.** EMA loss with $\alpha = 0.99$. Restart triggers when $|\Delta\bar{\mathcal{L}}| < 10^{-4}$ for 50 epochs.

# B Reproducibility

Code available at: `[GitHub URL to be added]`

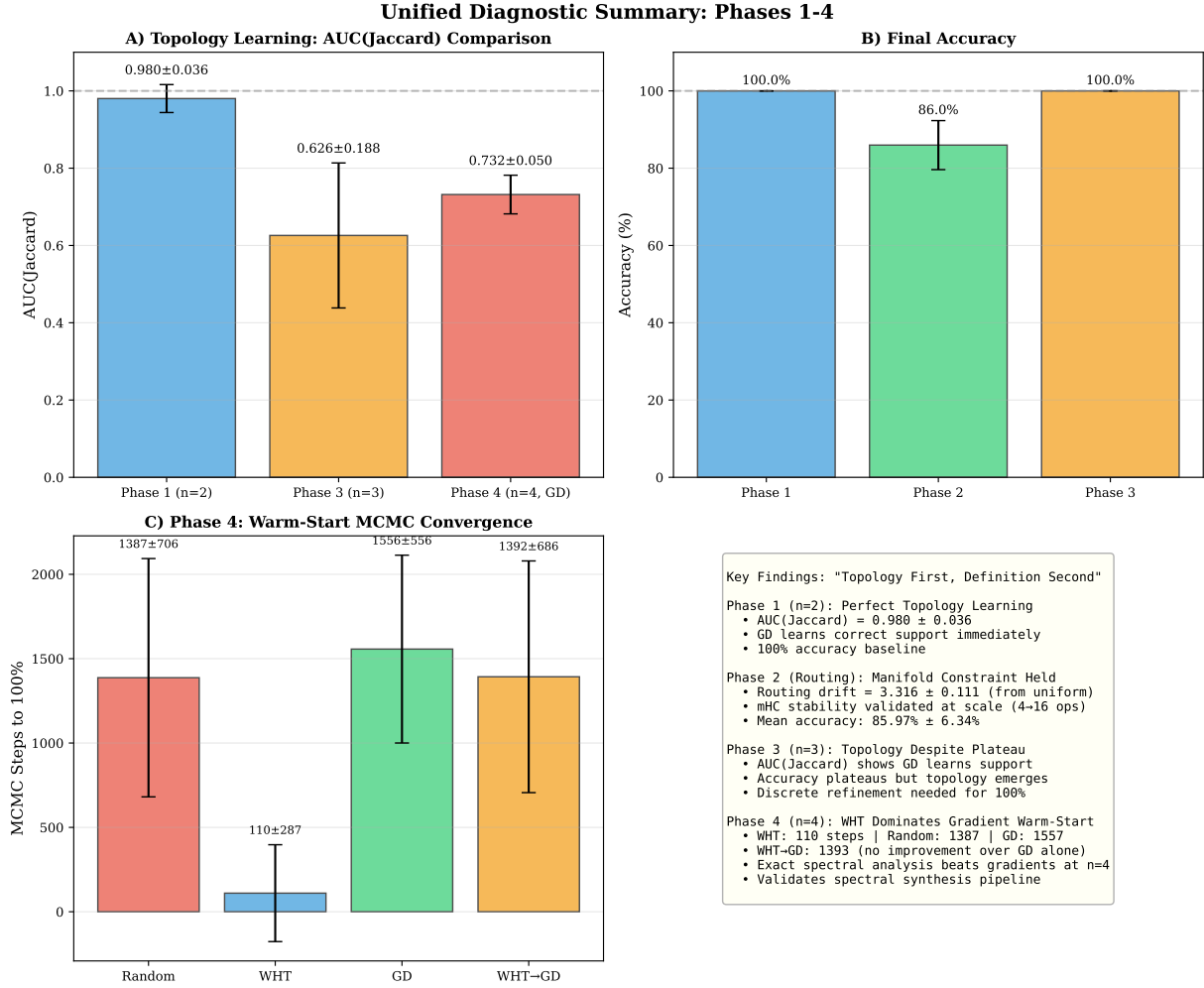Hardware: NVIDIA RTX 5060 (8GB VRAM), Intel Ultra 5 225F, 64GB RAM.

Figure 10: **Unified Diagnostic Summary Across All Phases.** (A) AUC(Jaccard) comparison showing topology learning quality across phases. (B) Final accuracy across Phases 1-3. (C) Phase 4 warm-start MCMC convergence: WHT initialization achieves fastest convergence. (D) Key insights summary validating "topology first, definition second" thesis and showing where exact spectral methods (WHT) surpass gradient-based approaches.
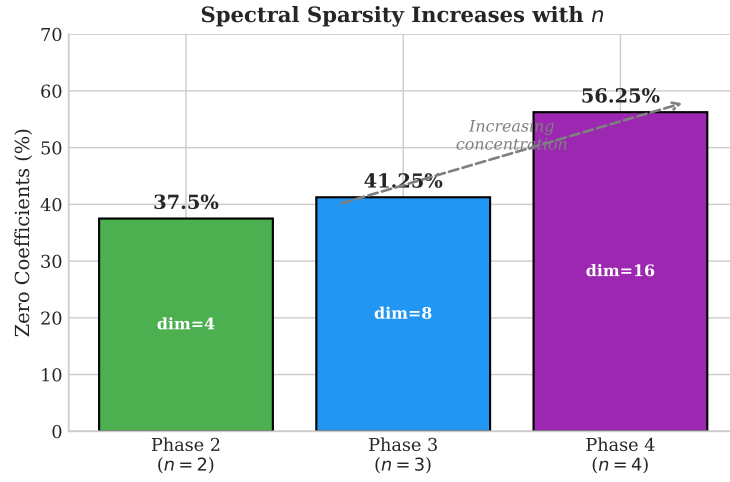
Figure 11: Spectral sparsity across phases. While sparsity varies by operation complexity ($31\% \rightarrow 39\% \rightarrow 36\%$), parity operations consistently achieve maximum sparsity (support=1) regardless of dimension. The variation reflects operation selection: Phase 4 includes dense symmetric functions (`and_4`, `or_4`) that require all 16 characters.