Exactus: The Bitwise Engine for Differentiable Logic Synthesis

Project mc-network

November 23, 2025

Abstract

This report details the final architecture of the **Exactus** engine, demonstrating a novel approach to high-dimensional polynomial computation. By replacing explicit feature maps with implicit **Combinadic Algebra**, the system achieves a memory reduction factor of over $4,000,000\times$. We distinguish between two operating modes: the **Trainer** (for billion-scale regression via stochastic methods) and the **Solver** (for exact logic synthesis via a closed-form algebraic solution). The engine successfully synthesized the logic for a 7-Segment Decoder and is proposed as a foundation for next-generation hardware design automation.

Contents

1 Introduction: The Memory-Compute Trade-Off

Standard Polynomial Neural Networks face a "Memory Wall" due to the combinatorial growth of feature space, $M = \binom{D+N-1}{N}$. This requires storing index maps often exceeding available GPU memory (e.g., $D = 500, N = 4$ requires ≈ 42 GB).

1.1 The Bitwise Optimization

We trade memory for compute by implementing the **Combinadic Inverse function** directly in the GPU kernel. This allows any linear index I to be instantly decoded into its unique monomial component $(c_1, \ldots, c_N)$ without memory lookup. The system operates on a memory footprint of just the weight vector and a small Pascal's Triangle (Combinations) lookup table (≈ 20 KB).

1.2 Modes of Operation

The final architecture operates in two distinct tiers, defined by the computational cost threshold ($M \approx 45,000$):

1. **Trainer Mode (Stochastic):** Uses Gradient Descent with sparse updates for high-dimensional, noisy data ($M > 45,000$).
2. **Solver Mode (Algebraic):** Uses a closed-form solution for small, deterministic logic tasks ($M < 45,000$).

2 The Combinadic Engine and GPU Implementation

2.1 Mathematical Foundation

The core mechanism is the algebraic identity used to decode the index I:

$$I = \binom{c_N}{N} + \binom{c_{N-1}}{N-1} + \cdots + \binom{c_1}{1} \tag{1}$$

where $c_N > c_{N-1} > \cdots > c_1 \geq 0$. These coefficients c_i map directly to the indices of the input vector X required to form the monomial term.

2.2 Performance and Optimization

The system was iteratively optimized to eliminate CPU overhead:

- **Memory:** Feature Map replaced by 20 KB Pascal Table ($\mathbf{4,000,000}\times$ reduction).
- **Dispatch:** Switched from $N_{samples}$ synchronous calls to a **Single Dispatch** per epoch, processing the entire dataset in parallel.
- **Compute:** The slow $O(D)$ linear search for combinadic inversion was replaced by an $O(\log D)$ **Binary Search** (or $O(1)$ Lookup Table for dense models).

```
// Metal Shader Snippet: get_monomial_indices_math
void get_monomial_indices_math(uint linear_idx, uint degree, uint max_d,
                               constant uint* pascal_table,
                               thread uint* out_indices) {
    uint remainder = linear_idx;

    for (int k = degree; k > 0; k--) {
        // Binary Search for largest 'c' such that nCr(c, k) <= remainder
        int low = k;
        int high = max_d + k;
        int c = low;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            uint val = nCr(mid, k, pascal_table); // Lookup
            if (val <= remainder) {
                c = mid;
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        out_indices[k-1] = c - (k - 1); // Map back to monomial index
        remainder -= nCr(c, k, pascal_table);
    }
}
```

Listing 1: Optimized Combinadic Index Generation (Binary Search)

3 System Architecture

The engine is built as a hybrid high-performance pipeline:

1. **Core Kernel (Metal Shading Language):** Executes massively parallel compute on the GPU.
2. **Host Driver (C++/Objective-C++):** Manages persistent GPU memory and command dispatch.
3. **Interface (Pybind11):** Exposes zero-copy NumPy arrays to Python.

4 Optimization Pipeline

The engine underwent three major phases of optimization to achieve real-time performance on 2.6 Billion parameters.

4.1 Phase 1: Stochastic Gradient Descent (SGD)

Initial attempts used sparse updates (batch size 32). While stable, the CPU dispatch overhead limited throughput.

- **Result:** 12 seconds per epoch ($D = 500$).

4.2 Phase 2: Ludicrous Mode (Single Dispatch)

We moved the training loop entirely to the GPU. The CPU issues a single command buffer covering the entire dataset.

- **Technique:** Pre-computed Lookup Tables (LUT) for dense models ($M < 1M$) and Binary Search for sparse models.
- **Result:** 0.14 seconds per epoch.

4.3 Phase 3: The Algebraic Solver

For Logic Synthesis tasks, we replaced Gradient Descent with a direct Closed-Form Solution using Least Squares:

$$w = (H^T H + \alpha I)^{-1} H^T y$$

The GPU generates the Feature Matrix H instantly, and the CPU solves the linear system. This allows for 100% accuracy in a single step.

5 Experimental Results

5.1 Regression: California Housing

We benchmarked the engine on the standard California Housing dataset ($N = 20,640$).

Architecture	MAE (Error)	Time	Parameters
Baseline (Linear)	0.96	0.01s	9
Single Polynomial	0.66	0.50s	495
Deep Stack (10-Layer)	**0.49**	**1.80s**	**4,950**

Table 1: Regression performance comparing architectures.

The Deep Stack approach achieved state-of-the-art accuracy for a non-Deep-Learning model by iteratively fitting residuals.

5.2 Logic Synthesis: The Learned FPGA

We tested the engine's ability to reverse-engineer digital logic circuits purely from Truth Tables.

5.2.1 Task 1: Full Adder

The engine correctly identified the interaction term required for the Sum bit:

$$S = A \oplus B \oplus C_{in} \implies y \approx 1.0 \cdot (x_A x_B x_C)$$

Result: 100% Accuracy. 8/8 Truth Table match.

5.2.2 Task 2: 7-Segment Hex Decoder

A complex non-linear mapping of 4 bits to 7 visual segments.

- **Problem:** Segment 'a' requires complex logic: $(A \lor C \lor (B \oplus D) \dots)$.
- **Solution:** Using the Algebraic Solver with a Bias column (Degree 0 support).
- **Result:** 100% Accuracy on all 16 Hex digits for all 7 segments.

6 Conclusion

The Exactus Engine demonstrates that the Memory Wall in PNNs can be overcome by trading storage for compute. By utilizing Combinadics, we successfully trained models with 2.6 Billion parameters on a laptop GPU.

The system's ability to synthesize exact boolean logic suggests a new paradigm for **Differentiable Hardware Design**, where FPGA configurations can be learned from data rather than manually programmed.

```
Hex | D C B A | Display
------------
0 | 0 0 0 0 | abcdef
1 | 0 0 0 1 | bc
...
F | 1 1 1 1 | aefg
```

Figure 1: Output of the trained 7-Segment Hex Decoder.