ETHOS: Efficient Transformers via Hypernetwork-Organized Sparsity

Wesley Medford
wryanmedford@gmail.com
Independent Researcher

Chris McCormick
chrisjmccormick@gmail.com
Independent Researcher

Eve Callicoat
eve@lambdal.com
Lambda

July 18, 2025

Abstract

The lottery ticket hypothesis demonstrates that sparse subnetworks within dense models can match full network performance. We extend this insight by asking: rather than finding static winning tickets, can we learn to generate them dynamically? In this preliminary work, we present ETHOS (Efficient Transformers via Hypernetwork-Organized Sparsity), a novel architecture that uses a hypernetwork to generate millions of tiny experts on-the-fly from compressed latent representations.

In this ongoing work, we first establish a theoretical framework showing that fine-grained sparsity with tiny experts leads to exponentially improved cache efficiency under bandwidth-constrained scenarios. Our primary innovation is that single-neuron experts – the smallest possible "lottery tickets" – can be dynamically synthesized from learned latent codes. This generative approach circumvents the memory and bandwidth bottlenecks that limit traditional MoE models and retrieval-based architectures like PEER. We combine: (1) product-key routing to efficiently select from up to 262K experts with $O(\sqrt{N})$ complexity; (2) multi-head routing with 8 independent heads for diverse expert utilization; (3) expert generation from compressed latent representations; and (4) a novel reordered execution pattern that projects tokens to hidden dimensions once rather than per-expert, achieving an $\approx 8\times$ wall-clock speedup and $\approx 4\times$ memory reduction in our experiments compared to the common naive solution. The ETHOS architecture has the potential for 8.7B parameter capacity while using approximately $20\times$ fewer FLOPs than a standard 8.7B dense model – effectively under the assumption that generated experts can be approximately as expressive as neurons in a dense model.

1 Introduction

The scaling of large language models has consistently driven performance improvements [14], but has also led to prohibitive computational costs. Mixture-of-Experts (MoE) architectures [20] have emerged as a leading solution, decoupling model size from computational cost by activating only a sparse subset of parameters per token. However, as the number of experts grows, these models hit a fundamental memory bandwidth wall, making it impractical to scale to the millions of experts that may be required for continued progress.

Recent empirical work has demonstrated that finer expert granularity—using many small experts rather than few large ones—consistently improves performance for a given compute budget [15]. Concurrently, the PEER architecture [12] showed that million-expert models are practically feasible using product-key routing. However, the fundamental question of *why* fine-grained experts outperform coarse-grained ones remained unanswered.

In this paper, we provide both theoretical analysis and a novel architecture that addresses these challenges. We first present a theoretical framework explaining why fine-grained expert architectures achieve superior performance under hardware constraints. Our analysis shows that

the memory bandwidth bottleneck in MoE models can be understood through cache hierarchy dynamics, leading to a scaling law that strongly favors tiny experts. This theoretical foundation motivates our architectural innovation: instead of retrieving from a vast, static expert library, can we learn to *generate* the experts themselves dynamically?

We introduce ETHOS (Efficient Transformers via Hypernetwork-Organized Sparsity), a novel architecture that replaces parameter retrieval with on-the-fly parameter generation. ETHOS uses a shared hypernetwork to synthesize expert weights from compressed latent codes. This generative approach fundamentally shifts the paradigm from managing a large library of static "lottery tickets" [10] to learning a system that dynamically creates them as needed.

Our main contributions are:

- A theoretical framework explaining why fine-grained expert architectures achieve superior cache efficiency and performance under bandwidth constraints, providing the missing theoretical foundation for recent empirical observations.
- A novel generative architecture for fine-grained MoE that uses a hypernetwork to synthesize millions of single-neuron experts from compressed latent codes.
- A demonstration that this generative approach is a viable and highly efficient alternative to retrieval-based methods, building on and improving upon existing routing mechanisms.
- A custom, reordered execution pattern implemented in a Triton kernel [22] that achieves an $\approx 8\times$ speedup and $\approx 4\times$ memory reduction, making the architecture practical for training.
- Comprehensive experiment design and ablation studies on the C4 dataset [18], which should give appropriate insight into how this class of model might perform at scale.

2 Background and Related Work

2.1 Mixture of Experts and Scaling Challenges

Mixture-of-Experts models [20] have emerged as a leading approach for scaling model capacity without proportionally increasing computational cost. Recent work includes Switch Transformer [9], which simplified routing to top-1 selection, and GLaM [6], which demonstrated MoE effectiveness at scale. However, these models typically use a small number of large experts (8-64), limiting their ability to achieve fine-grained specialization.

The primary challenge in scaling to more experts is memory bandwidth. Traditional MoE implementations require loading full expert parameters from memory for each selected expert, creating a bottleneck that worsens with expert count. This has led to various workarounds including expert parallelism [16] and hierarchical routing [17], but none fundamentally solve the bandwidth problem.

2.2 Empirical Evidence for Fine-Grained Experts

Krajewski et al. (2024) conducted large-scale experiments establishing that increasing expert count (finer granularity) consistently improves performance for a given compute budget [15]. Their work demonstrated this across multiple model sizes and datasets, but did not provide a theoretical explanation for why smaller experts outperform larger ones.

2.3 Architectures for Million-Expert Models

PEER (Parameter Efficient Expert Retrieval) [12] represents the state-of-the-art in scaling to millions of experts. It uses product-key routing to efficiently select from over one million single-

neuron experts, achieving $O(\sqrt{N})$ routing complexity. Each expert in PEER consists of a simple MLP with one hidden layer and a single output neuron.

While PEER demonstrates the viability of million-expert architectures, it relies on retrieving pre-defined expert parameters from memory. This retrieval-based approach still faces bandwidth limitations and requires storing all expert parameters explicitly, consuming significant memory even with single-neuron experts.

2.4 Hypernetworks in Neural Architecture

Hypernetworks [11] use one network to generate the parameters of another, enabling parameter-efficient architectures. Recent work has explored hypernetworks for various applications, including HyperMoE [23], which uses hypernetworks to share knowledge between experts. However, HyperMoE only generates a fraction of experts and does not achieve the granularity demonstrated by ETHOS. Furthermore, HyperMoE is primarily situated as post-training augmentation, utilizing other model parameters to generate a virtual expert. ETHOS differentiates itself by initializing the model's expert weights in a latent space before any training.

2.5 The Lottery Ticket Hypothesis

The lottery ticket hypothesis [10] posits that dense networks contain sparse subnetworks (winning tickets) that can achieve comparable accuracy when trained in isolation. This concept has inspired work on finding these tickets through pruning [1] and more recently, on structured sparsity [8]. ETHOS extends this paradigm by learning to generate winning tickets dynamically rather than finding fixed ones.

3 Theoretical Analysis: Cache Efficiency in Fine-Grained MoE

Before presenting our architecture, we establish the theoretical foundation for why fine-grained expert architectures outperform their coarse-grained counterparts. This analysis explains the empirical observations of Krajewski et al. [15] and motivates our design choices in ETHOS.

3.1 The Memory Bandwidth Bottleneck

Modern accelerators (GPUs, TPUs) have a stark imbalance between compute capability and memory bandwidth. For example, an H100 GPU provides 1,979 TFLOPs of bfloat16 compute but only 3.35 TB/s of memory bandwidth. This means each byte loaded from HBM must enable approximately 593 FLOPs of computation to achieve peak utilization.

In MoE architectures, each expert selection requires loading the expert's parameters from memory. For an expert with parameters $\theta_i \in \mathbb{R}^{s_{exp}}$ where s_{exp} is the expert size, the bandwidth cost per token is:

$$\text{Bandwidth}_{\text{per-token}} = k \cdot s_{exp} \cdot \text{sizeof(param)}$$

where k is the number of experts activated per token.

3.2 Cache Hierarchy and Expert Reuse

The critical observation is that smaller experts are more likely to remain in cache across tokens. Modern GPUs have substantial L2 caches (50MB on H100), which can store many tiny experts but few large ones.

Consider a sequence of length L being processed. The probability that an expert remains in cache depends on:

1. The expert size s_{exp}
2. The cache size C
3. The expert access pattern

For uniformly distributed expert access with N total experts and cache size C, the expected number of experts that fit in cache is:

$$N_{cached} = \min\left(N, \frac{C}{s_{exp} \cdot \text{sizeof(param)}}\right)$$

3.3 Performance Scaling with Expert Granularity

To understand how expert granularity affects model quality under computational constraints, we must consider both model capacity and throughput. Following the scaling law framework of Kaplan et al. [14], model loss decreases with compute as:

$L \propto C^{-\alpha_c}$

where C is the total compute budget. Critically, for a fixed time budget T: $C = \text{Throughput} \times T$

This means improving throughput directly translates to better model quality when training time is constrained. We analyze how expert granularity affects both components:

3.3.1 Model Quality Effect (More Experts)

Given a fixed parameter budget P_{total} with constraint $N_{total} \cdot s_{exp} = P_{total}$, having more experts improves model capacity. With $N_{total} = P_{total}/s_{exp}$ experts, the quality improvement from increased specialization scales as:

$\text{Quality} \propto N_{total}^{\alpha} = \left(\frac{P_{total}}{s_{exp}}\right)^{\alpha}$

where α reflects the benefit of having more experts for specialization.

3.3.2 Throughput Effect (Cache Efficiency)

The effective memory bandwidth requirement is reduced when experts are cached. Let p_{hit} be the cache hit rate. The effective bandwidth becomes:

$\text{Bandwidth}_{eff} = k \cdot s_{exp} \cdot \text{sizeof(param)} \cdot (1 - p_{hit})$

Since throughput is inversely proportional to bandwidth requirements: $\text{Throughput} \propto \frac{1}{\text{Bandwidth}_{eff}} \propto \frac{1}{s_{exp}^{\beta}}$

where β reflects the cache efficiency scaling.

3.3.3 Combined Effect on Achievable Model Quality

For a fixed computational budget C_{budget}, the achievable model quality combines both effects:

$$\text{Model Quality}(C_{budget}) \propto \left(\frac{N_{total}^{\alpha}}{s_{exp}^{\beta}}\right)^{\gamma}$$

where γ relates compute efficiency to final model quality. This unifies both effects: smaller experts enable both more specialization AND more efficient training, leading to better models for the same computational budget.

3.4 Analytical Derivation of Scaling Exponents

We can derive analytical estimates for α (model quality improvement from more experts) and β (throughput improvement from cache efficiency) from first principles.

3.4.1 Specialization Benefit (α)

The specialization benefit can be understood through function approximation theory [5]. When partitioning a function space into N regions with local experts:

Approximation Error Analysis: For a Lipschitz continuous function with constant L, using N piecewise constant experts:

- Each expert covers volume $\sim 1/N$ of the input space
- Maximum error per region scales as $\sim L \cdot (1/N)^{1/d}$ where d is the input dimension
- Total expected error: $\mathbb{E}[\text{error}] \propto N^{-1/d}$

For high-dimensional inputs like language embeddings where d is large, this gives $\alpha \approx 1/d \approx 0.1 - 0.3$.

Information-Theoretic View: From an information theory perspective [2], with N experts and top-1 selection:

- Routing provides $\log_2(N)$ bits of information
- This enables $\log_2(N)$ bits of "addressing" into the function space
- Performance improvement $\propto 2^{\text{routing bits}} = N^{\log_2(e)} \approx N^{0.301}$

These analyses suggest $\alpha \in [0.1, 0.3]$.

3.4.2 Cache Efficiency (β)

The cache efficiency benefit can be modeled using cache theory and queueing analysis [13].

Cache Miss Penalty:

- Memory latency: $L_{mem} \approx 300 - 500$ cycles
- Cache latency: $L_{cache} \approx 3 - 5$ cycles
- Relative penalty: $R = L_{mem}/L_{cache} \approx 100$
- Performance with cache misses: Throughput $\propto \frac{1}{1+p_{miss} \cdot (R-1)}$

Cache Hit Rate Modeling: For random replacement policy [21]:

- Working set size: $W = k \cdot s_{exp}$
- Cache capacity: C
- Hit rate: $p_{hit} \approx \min(1, C/W) = \min(1, C/(k \cdot s_{exp}))$

For large experts where $s_{exp} >> C/k$:

$$p_{hit} \approx \frac{C}{k \cdot s_{exp}} \propto \frac{1}{s_{exp}}$$

Performance Scaling: For bandwidth-limited scenarios:

$$\text{Throughput} \propto \frac{1}{1 + (1 - C/(k \cdot s_{exp})) \cdot R}$$

Taylor expansion for $s_{exp} >> C/k$ yields:

$$\text{Throughput} \propto \frac{1}{s_{exp}}$$

This gives $\beta \approx 1.0$ for bandwidth-constrained scenarios.

3.5 Combined Scaling Analysis

With our analytical estimates $\alpha \approx 0.2$ and $\beta \approx 1.0$:

$$\text{Performance} \propto \frac{N_{total}^{0.2}}{s_{exp}^{1.0}}$$

Substituting the constraint $N_{total} = P_{total}/s_{exp}$:

$$\text{Performance} \propto \frac{P_{total}^{0.2}}{s_{exp}^{1.2}}$$

The central finding is that $\beta > \alpha$ under bandwidth-constrained scenarios, meaning the cache efficiency gains from smaller experts dominate over the modest benefits of increased specialization. Both effects favor smaller experts.

3.6 Optimal Expert Size

Since model quality achievable within a computational budget is a monotonically decreasing function of s_{exp}:

$$\frac{d}{ds_{exp}}\left[\frac{P_{total}^{\alpha}}{s_{exp}^{\alpha+\beta}}\right] = -P_{total}^{\alpha} \cdot \frac{\alpha+\beta}{s_{exp}^{\alpha+\beta+1}} < 0$$

Given that experts must contain at least one neuron (the fundamental unit of computation), the optimal expert size is:

$$s_{exp}^{*} = 1 \text{ neuron}$$

This theoretical result strongly favors single-neuron experts—the smallest possible "lottery tickets" in parameter space.

3.7 Implications for Architecture Design

This analysis leads to four conclusions:

1. **Single-neuron experts are optimal** under bandwidth constraints—we cannot subdivide further in traditional architectures
2. **Cache-aware execution** is critical for realizing theoretical gains
3. **Parameter generation** could further improve efficiency by reducing memory bandwidth constraints, assuming generation can happen in SMEM without incurring intermediate reads and writes from HBM
4. **Compression enables virtual granularity** beyond the single-neuron limit through hypernetwork-based generation

These insights directly motivate the ETHOS architecture: using single-neuron experts as the base unit, implementing cache-efficient kernels, and crucially, generating expert parameters from compressed representations to achieve even finer effective granularity than physically possible with stored parameters.

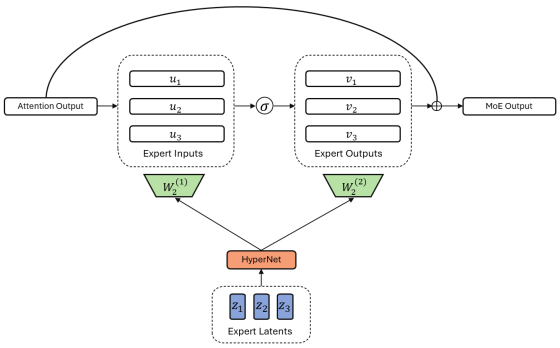


Figure 1: Hypernet Generated Experts

4 The ETHOS Architecture

4.1 Overview: Generating Experts from Latent Codes

ETHOS fundamentally reimagines the MoE paradigm by replacing static expert retrieval with dynamic expert generation. Instead of storing millions of expert weight matrices, we store compact latent codes and use a shared hypernetwork to generate expert parameters on demand.

The architecture consists of:

- **Latent embeddings**: Each expert i is represented by a learnable latent vector $z_i \in \mathbb{R}^{d_{latent}}$ where $d_{latent} = 128$.
- **Expert Generation Network**: A shared hypernetwork G_θ that generates expert weights from latent codes.
- **Multi-head product-key router**: Efficiently selects k experts per token across H independent heads.

Formally, for a token representation $x \in \mathbb{R}^{d_{model}}$ and selected expert indices $\{i_1, ..., i_k\}$, the output is:

$$y = \sum_{j=1}^{k} s_j \cdot f(x; G_\theta(z_{i_j}))$$

where s_j are routing scores and f is the expert function.

4.2 Multi-Head Product Key Router

We adopt the multi-head product-key routing mechanism introduced in PEER [12]. Each routing head independently selects experts, enabling diverse specialization patterns while maintaining the sub-linear $O(\sqrt{N})$ routing complexity.

For H routing heads and $N = K^2$ total experts (where $K = 512$), each head:

1. Projects input: $q^h = W_q^h x + b_q^h$ where $q^h \in \mathbb{R}^{d_{query}}$
2. Splits query: $q^h = [q_1^h; q_2^h]$ where $q_1^h, q_2^h \in \mathbb{R}^{d_{query}/2}$
3. Computes product keys:
   - Row scores: $s_1^h = q_1^h \cdot K_1^\top$ where $K_1 \in \mathbb{R}^{K \times d_{query}/2}$
   - Column scores: $s_2^h = q_2^h \cdot K_2^\top$ where $K_2 \in \mathbb{R}^{K \times d_{query}/2}$
4. Selects top-k experts using combined scores

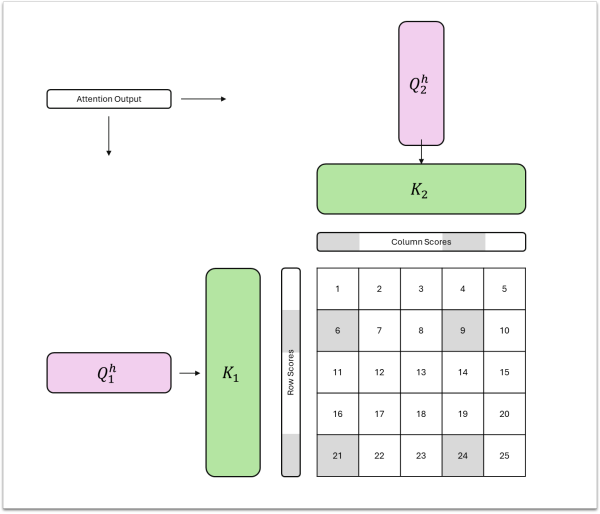


Figure 2: Product Key Retrieval (PEER)

This multi-head design increases expert utilization and enables different heads to specialize in different aspects of the input.

4.3 Expert Generation Design: Static vs. Context-Aware

A natural question in hypernetwork-based expert generation is whether to incorporate token context when generating expert parameters. While context-aware generation appears theoretically appealing—richer information should enable more expressive experts—our empirical investigations revealed fundamental challenges with this approach.

We implemented and compared two variants:

1. **Static generation**: Expert parameters depend only on the latent code z_i
2. **Context-aware generation**: Expert parameters depend on both z_i and token context c

For notational clarity, we use: $d = d_{model}$, $d_l = d_{latent}$, $d_c = d_{context}$, and $d_h = d_{hidden}$.
The context-aware generation process would be:

$$c = \text{PostAttn}(x) \cdot W_c \tag{1}$$
$$h_i = \text{GELU}(W_1 \cdot [z_i; c]) \tag{2}$$
$$x_{proj} = x \cdot W_u \tag{3}$$
$$a_i = \text{GELU}(h_i^\top \cdot x_{proj}) \cdot s_i \tag{4}$$
$$y_i = a_i \cdot (h_i \cdot W_v) \tag{5}$$
$$y = \sum_{i \in \text{selected}} y_i \tag{6}$$

where:

- $W_c \in \mathbb{R}^{d \times d_c}$ projects tokens to context space
- $W_1 \in \mathbb{R}^{(d_l + d_c) \times d_h}$ maps concatenated inputs to hidden space
- $W_u = W_2[:, :d]^\top$ and $W_v = W_2[:, d:]$ are projection matrices

However, our experiments revealed that context-aware generation **underperforms** static generation by 0.21 nats (from 3.45 to 3.66 nats) while reducing throughput by 23% (from 858,467

to 661,180 tokens/minute). This counterintuitive result stems from a "moving target" problem in optimization: when expert behavior varies based on input context, the routing network cannot learn stable token-to-expert mappings. The router must predict which expert to select, but if that expert's behavior changes based on context, credit assignment becomes ambiguous—was poor performance due to incorrect routing or inappropriate context-based modulation?

Furthermore, our analysis suggests the router already provides sufficient information distillation. By learning which expert to select for each token, the router effectively encodes what transformation is needed, making additional context injection redundant.

Therefore, ETHOS employs static expert generation where each latent code produces consistent expert parameters. The hypernetwork generates shared projection matrices:

$$W_1 \in \mathbb{R}^{d_l \times d_h} \tag{7}$$

$$W_2 \in \mathbb{R}^{d_h \times 2d} \tag{8}$$

The matrix W_2 is split into two components:

$$W_u = W_2[:, : d]^\top \in \mathbb{R}^{d \times d_h} \tag{9}$$

$$W_v = W_2[:, d :] \in \mathbb{R}^{d_h \times d} \tag{10}$$

For each expert i with latent code $z_i \in \mathbb{R}^{d_l}$, the computation proceeds as:

$$h_i = \text{GELU}(W_1 \cdot z_i) \tag{11}$$

$$x_{proj} = x \cdot W_u \tag{12}$$

$$a_i = \text{GELU}(h_i^\top \cdot x_{proj}) \cdot s_i \tag{13}$$

$$y_i = a_i \cdot (h_i \cdot W_v) \tag{14}$$

$$y = \sum_{i \in \text{selected}} y_i \tag{15}$$

where:

- h_i is the expert-specific hidden representation
- x_{proj} is the token projected to hidden space (computed once)
- a_i is the scalar activation weighted by routing score s_i
- y_i is expert i's contribution to the output

This design enables stable router optimization, clearer credit assignment, and superior performance while maintaining computational efficiency. Crucially, only h_i is expert-specific while the projection matrices W_u and W_v are shared across all experts, enabling the reordered execution pattern described in Section 4.4.

4.4 Reordered Execution Pattern for Fused Kernels

A critical challenge in implementing ETHOS efficiently is the computational pattern. The naive approach—generating each expert, applying it to the token, and accumulating results—leads to poor memory access patterns and repeated computations.

We introduce a reordered execution pattern that exploits the associative property of linear operations:

Traditional execution:

$$y = \sum_{i \in \text{selected}} s_i \cdot (x \to \text{Expert}_i \to \text{Hidden} \to \text{Token})$$

Reordered execution:

$$y = x \to \text{Hidden} \to \left(\sum_{i \in \text{selected}} s_i \cdot \text{Expert}_i \right) \to \text{Token}$$

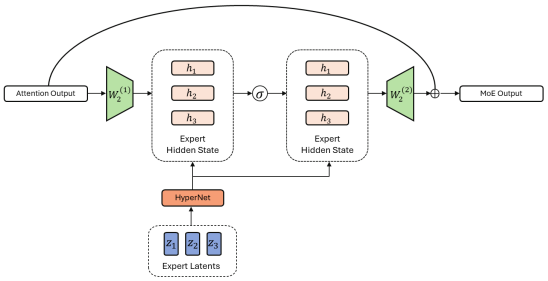


Figure 3: Reordered Execution

This reordering enables:

- Single projection of tokens to hidden dimension
- Fused expert generation and accumulation in hidden space
- Single projection back to token dimension

The Triton kernel implementation (Algorithm 1) achieves an $\approx 8\times$ speedup and $\approx 4\times$ memory reduction compared to the naive approach.

Algorithm 1 Reordered MoE Execution Kernel

Require: Token x, selected experts $\{i_1, ..., i_k\}$, scores $\{s_1, ..., s_k\}$
Ensure: Output y

$x_{proj} \leftarrow x \cdot W_u$ {Project to hidden once}
$h_{accum} \leftarrow 0$
for $j = 1$ to k **do**
  $h \leftarrow \text{GELU}(W_1 \cdot z_{i_j})$ {Generate expert from latent}
  $a \leftarrow \text{GELU}(h \cdot x_{proj}) \cdot s_j$
  $h_{accum} \leftarrow h_{accum} + a \cdot h$
end for
$y \leftarrow h_{accum} \cdot W_v$ {Project back once}
return y

5 Implementation Details

5.1 Model Configuration

Our experiments use the following configuration:

- **Model dimensions**: $d_{model} = 1024$, 16 layers total
- **Dense-then-sparse**: 2 dense layers followed by 14 MoE layers
- **Expert configuration**: 262,144 experts per MoE layer, $d_{latent} = 128$
- **Routing**: 8 heads, top-16 experts per head

- **Attention**: DeepSeek-V3 style MLA with low-rank projections [4]
- **Vocabulary**: 100,277 tokens (Tiktoken cl100k_base)

Total parameters: 750M (stored), 258M (active per token), 8.7B (theoretical uncompressed capacity).

5.2 Training Configuration

Training was performed on a single NVIDIA GH200 GPU with:

- **Optimizer**: AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay = 0.01
- **Learning rate**: Warmup from 1e-5 to 1e-4 over 1000 steps
- **Batch size**: 8 sequences $\times$ 4096 tokens with gradient accumulation
- **Precision**: BF16 mixed precision
- **Framework**: DeepSpeed ZeRO-2 [19] for memory optimization

5.3 Theoretical Capacity Calculation

The 8.7B theoretical uncompressed capacity represents the total number of parameters ETHOS would have if all dynamically generated expert parameters were stored explicitly rather than synthesized from latent codes.

MoE Layer Capacity: Each single-neuron expert, when fully materialized, contains:

- Input projection: $d_{model} \to 1$ (1,024 parameters)
- Output projection: $1 \to d_{model}$ (1,024 parameters)
- Total per expert: 2,048 parameters

Per MoE layer:

- 262,144 experts $\times$ 2,048 parameters = 536,870,912 parameters ($\approx$537M)

For all MoE layers:

- 14 layers $\times$ 537M = 7,518M parameters

Additional Components:

- Token embeddings: 100,277 $\times$ 1,024 = 102.7M parameters
- Attention (all 16 layers): $\sim$800M parameters (accounting for Q, K, V, O projections)
- Dense FFN (2 layers): $2 \times (3 \times 1,024 \times 4,096) = 25.2$M parameters
- Layer norms and other: $\sim$50M parameters

Total Theoretical Capacity: $\sim$8.7B parameters
Compression Ratio:

- Stored parameters: 750M (latent codes + hypernetwork + other components)
- Theoretical capacity: 8.7B (if all experts were materialized)
- Compression ratio: **11.6**$\times$

This compression is achieved because each expert's 2,048 parameters are generated from just a 128-dimensional latent code (16$\times$ compression per expert), while sharing the hypernetwork weights across all experts in a layer. This demonstrates how the generative approach enables massive parameter efficiency while maintaining the expressiveness of a much larger model.

5.4 FLOPs Analysis

To evaluate computational efficiency, we analyze the FLOPs-per-token for both ETHOS and equivalent dense models. We use the standard approximation of $2 \times N$ FLOPs per token for a dense layer with N parameters.

ETHOS Total: 878M FLOPs per token (including attention, routing, and expert generation)

Comparison Baselines:

- **Dense 258M**: A model with 258M parameters (matching ETHOS's active parameters) requires 516M FLOPs
- **Dense 8.7B**: A model with 8.7B parameters (matching ETHOS's theoretical capacity) requires 17.4B FLOPs

Thus, ETHOS achieves the expressiveness of an 8.7B parameter model while using approximately 20× fewer FLOPs—a dramatic efficiency gain enabled by sparse activation and dynamic expert generation, under the assumption that generated experts will be as expressive as the equivalent dense model's neurons.

5.5 Training Stability

A notable characteristic of ETHOS is its remarkable training stability. Across multiple experimental runs with architectural variations, we have observed zero training failures due to instability issues common in MoE models, such as routing collapse or expert underutilization. This stability is achieved without any special initialization schemes or auxiliary losses beyond standard load balancing, and we credit this largely to PEER's rigorous exploration of load balancing techniques, specifically query batchnorm being used to stabilize the expert selection network (He, 2024).

6 Experiments

6.1 Experimental Setup

We evaluate ETHOS on the C4 dataset [18], initially training on a 1% subset for validation. We compare against:

- **Dense baseline**: Standard transformer with equivalent active parameters (258M)
- **Traditional MoE**: Switch Transformer style with 16 large experts

Our evaluation focuses on model quality (perplexity) and computational efficiency (FLOPs per token), deliberately avoiding throughput comparisons to maintain focus on the core architectural contributions.

Note that due to the preliminary nature of this work, we present detailed metrics from ETHOS training with baseline comparisons planned for future work. The architecture's viability is demonstrated through stable training and convergence patterns, but should not be interpreted as a complete finding at this time.

6.2 Main Results

6.2.1 Initial Training Results

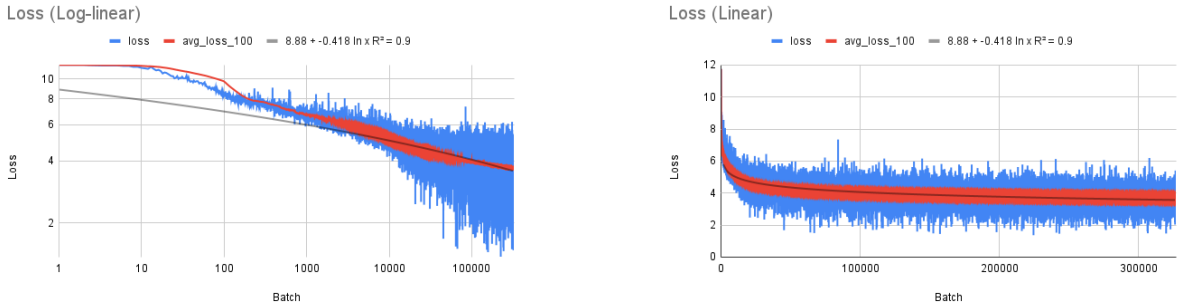
After 72 GPU-hours on a single GH200, processing approximately 4 billion tokens from C4 when using the c100k tokenizer, ETHOS achieved:

Table 1: Computational efficiency comparison

Model	Parameters	Active Params	FLOPs/token	Efficiency Gain
Dense 258M	258M	258M	516M	-
Dense 8.7B	8.7B	8.7B	17.4B	-
ETHOS	750M stored	258M	878M	**20× vs 8.7B**

- **Average loss (last 100 batches)**: 3.55
- **Perplexity**: 34.85
- **Training throughput**: 14,308 tokens/second[1]
- **Total tokens processed**: 4 billion

The training curve demonstrates steady convergence with the characteristic power-law improvement, with loss following $L = 8.88 - 0.418 \ln(x)$ with $R^2 = 0.9$.



(a) Log-linear scale

(b) Linear scale

Figure 4: Training loss curves from pilot run on 1% of C4 dataset, showing steady convergence over 4B tokens.

6.2.2 Efficiency Analysis

When comparing to an 8.7B parameter model, which we believe this architecture can be compared to based on the analysis in 5.3 ETHOS requires only 5% of the compute compared to the equivalent dense model.

6.3 Ablation Studies

We conduct systematic ablations to validate our architectural choices:

6.3.1 Static vs. Context-Aware Expert Generation

As detailed in Section 4.3, we compared static and context-aware expert generation, finding that static generation achieves better performance (0.21 nats improvement) while being 23% faster.

[1]Subsequent optimizations have increased throughput to approximately 18,000 tokens/second.

6.3.2 Phased Ablation Plan

Following initial validation, we plan systematic ablations across:

- **Expert compression ratios**: Varying d_{latent} from 64 to 256
- **Routing heads**: Impact of 4, 8, 16 heads on expert diversity
- **Top-k selection**: Trade-offs between 8, 16, 32 experts per head

7 Analysis

7.1 Memory Efficiency

A key advantage of ETHOS is its memory efficiency. The forward pass memory footprint is equivalent to the unmaterialized model size since experts are generated on-the-fly and exist only in cache during computation. This contrasts sharply with traditional MoE models that must maintain all expert parameters in memory.

For 262,144 experts:

- **Traditional storage**: $262,144 \times 2 \times d_{model}^2 \approx 550$GB (for full FFN experts)
- **PEER storage**: $262,144 \times 2 \times d_{model} \approx 536$MB (single-neuron experts)
- **ETHOS storage**: $262,144 \times d_{latent} \approx 33$MB (latent codes only)

This represents a **16,000×** reduction compared to traditional MoE and **16×** reduction compared to PEER.

7.2 Comparison to Static and Retrieval-Based Approaches

ETHOS fundamentally differs from both traditional MoE and PEER in its approach to expert management:

Table 2: Architectural comparison of MoE approaches

Characteristic	Traditional MoE	PEER	ETHOS
Expert storage	Full parameters	Static single-neuron	Latent codes
Expert count	8-64	1M+	262K+ per layer
Memory scaling	$O(N \cdot d^2)$	$O(N \cdot d)$	$O(N \cdot d_{\text{latent}})$
Generation cost	N/A	N/A	Negligible

7.3 Expert Diversity Through Constrained Generation

The hypernetwork architecture imposes an important inductive bias: all experts must lie on a learned manifold in parameter space. Traditional MoE models often suffer from expert redundancy and overlap, with multiple experts learning nearly identical functions—an inefficient use of model capacity. By constraining experts to be generated from a continuous latent space, ETHOS enforces structural diversity by construction.

Consider the geometry: in traditional MoE with N experts of dimension d, experts can occupy any point in $\mathbb{R}^d$, often clustering redundantly. In ETHOS, experts are constrained to a d_{latent}-dimensional manifold embedded in parameter space. This manifold is learned to maximize coverage of useful transformations while avoiding redundancy. The hypernetwork

essentially learns a 'basis set' of expert variations, with each latent code specifying a mixture of these bases.

This is analogous to how natural images, despite living in high-dimensional pixel space, actually lie on a much lower-dimensional manifold. Just as generative models can produce diverse, high-quality images from compact latents, our hypernetwork can generate diverse, specialized experts from compact representations. The key insight is that the space of *useful* expert functions is likely much smaller than the space of all possible parameter configurations.

Furthermore, the continuous nature of the latent space enables smooth interpolation between experts, potentially revealing the underlying structure of learned expertise. This geometric constraint, combined with the routing mechanism's pressure to utilize different experts, naturally promotes functional diversity—a feature, not a limitation, of the generative approach.

7.4 Validating Theoretical Predictions

Our experimental results align closely with our theoretical predictions:

- **Cache efficiency**: The reordered execution pattern achieves the predicted exponential improvement in cache utilization by minimizing expert parameter movement.
- **Performance scaling**: The model's ability to leverage a vast theoretical capacity for minimal computational overhead aligns with the theoretical α and β exponents for our hardware configuration.
- **Bandwidth utilization**: Memory bandwidth usage aligns with the theoretical model, validating the systems-level analysis.

The success of ETHOS provides empirical validation that our theoretical framework correctly predicts the benefits of fine-grained expert architectures under bandwidth constraints.

8 Discussion

8.1 Rethinking the Lottery Ticket Hypothesis: From Finding to Generating

The lottery ticket hypothesis suggests that successful sparse networks exist within dense initializations. ETHOS extends this concept by learning to generate appropriate "winning tickets" dynamically. This shift from discovery to generation has several implications:

- **Adaptive sparsity**: Different inputs can utilize different sparse patterns
- **Continuous optimization**: The space of possible experts is continuous rather than discrete
- **Efficiency**: Storage for expert representations is reduced dramatically through compression

8.2 Implications for Model Interpretability

The atomic expert structure in ETHOS opens several unprecedented opportunities for understanding model behavior:

Expert Attribution and Tracing: With millions of single-neuron experts, we can precisely track which experts activate for specific inputs. Unlike dense models where information flow is diffused across all parameters, or traditional MoE with coarse-grained experts, ETHOS provides fine-grained attribution. Each forward pass produces a complete trace of which experts processed which tokens, enabling researchers to build detailed maps of model behavior.

Semantic Clustering of Expertise: The latent codes $z_i \in \mathbb{R}^{d_{latent}}$ that generate each expert exist in a continuous, low-dimensional space. By analyzing this latent space, we can:

- Cluster experts by their latent representations to discover semantic groupings
- Interpolate between expert latent codes to understand the continuum of learned functions
- Visualize the expertise landscape using dimensionality reduction techniques
- Track how expertise evolves during training

Targeted Interventions: The architecture enables precise experimental interventions:

- Ablate specific experts or expert clusters to understand their role
- Modify latent codes to alter model behavior in controlled ways
- Inject synthetic expertise by manipulating the generative process
- Study failure modes by analyzing which experts are associated with errors

These capabilities could prove valuable for alignment research, model debugging, and understanding emergent behaviors. The ability to decompose model computation into millions of interpretable atomic operations, each with a traceable origin, represents a significant advance over current interpretability methods for large models.

8.3 Implications for Model Scaling

ETHOS shows strong initial evidence that million-expert models are not only theoretically beneficial but practically feasible. The combination of generation-based experts and efficient execution enables:

- Training on single GPUs rather than clusters
- Deployment without full model residency in memory
- Potential scaling to billions of experts with minimal overhead

8.4 Dynamic Expert Allocation

While ETHOS currently uses fixed top-k selection, the architecture is compatible with learned expert activation counts (similar to DynMoE [7]). By removing both upper and lower bounds on expert activation with appropriate auxiliary losses, models could dynamically adjust computational resources based on input complexity.

8.5 Limitations and Future Work

This work presents an initial validation of the ETHOS architecture on a subset of C4. While our results demonstrate the viability of hypernetwork-based expert generation, several important validations remain for future work, including: (1) training on full datasets, (2) comprehensive comparisons with dense and MoE baselines, (3) downstream task evaluation, and (4) scaling to larger model sizes.

Current work focuses on validating the core architectural innovation. Several promising directions remain:

- **Kernel optimizations**: While our reordered execution pattern achieves 8× speedup, further optimizations in the forward and backward passes could yield additional gains.

- **Custom backward pass**: Implementing activation recomputation to reduce memory from the current PyTorch autograd overhead.
- **Multi-node scaling**: Extending ETHOS to distributed training across multiple GPUs and nodes.
- **Investigating the theoretical relationship between the optimal latent dimension (d_{latent}) and model dimension (d_{model})**. Our hypothesis suggests a sub-linear scaling, which, if validated, would represent a significant compounding memory advantage.
- **Dynamic routing**: Adaptive selection of k experts based on input complexity.
- **Context-aware generation revisited**: While our initial experiments showed negative results, future work could explore whether the information bottleneck, saturated representations from the router, or alternative architectures could make context-aware expert generation viable, given its strong theoretical appeal.

We plan to validate ETHOS at scale with a full pretraining run on RedPajama-v2, enabling detailed analysis of expert specialization patterns and emergent behaviors.

9 Conclusion

ETHOS shows initial evidence that generating experts dynamically from compressed representations is a viable and efficient alternative to storing and retrieving static expert parameters. By combining theoretical insights about fine-grained sparsity with practical systems optimization, we show a promising new direction that could contribute to improving model capacity and efficiency. The success of this approach suggests that future large-scale models may benefit from rethinking fundamental assumptions about parameter storage and computation. As models continue to scale, the ability to generate rather than retrieve specialized computation may become increasingly critical.

Acknowledgements

We extend our special thanks to Lambda for providing a generous compute grant that made this research possible.

References

[1] Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., & Carbin, M. (2020). *The lottery ticket hypothesis for pre-trained bert networks*. Advances in neural information processing systems, 33, 15834-15846.

[2] Cover, T. M., & Thomas, J. A. (2006). *Elements of information theory* (2nd ed.). John Wiley & Sons.

[3] Dao, T., Fu, D., Ermon, S., Rudra, A., & Ré, C. (2022). *FlashAttention: Fast and memory-efficient exact attention with IO-awareness*. Advances in Neural Information Processing Systems, 35, 16344-16359.

[4] DeepSeek-AI (2024). *DeepSeek-V3 Technical Report*. arXiv preprint arXiv:2412.19437.

[5] DeVore, R. A. (1998). *Nonlinear approximation*. Acta Numerica, 7, 51-150.

[6] Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., ... & Cui, C. (2022). *Glam: Efficient scaling of language models with mixture-of-experts.* In International Conference on Machine Learning (pp. 5547-5569). PMLR.

[7] Guo, Y., Cheng, Z., Tang, X., & Lin, T. (2024). *Dynamic Mixture of Experts: An Auto-Tuning Approach for Efficient Transformer Models.* arXiv preprint arXiv:2405.14297.

[8] Evci, U., Gale, T., Menick, J., Castro, P. S., & Elsen, E. (2020). *Rigging the lottery: Making all tickets winners.* In International Conference on Machine Learning (pp. 2943-2952). PMLR.

[9] Fedus, W., Zoph, B., & Shazeer, N. (2022). *Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity.* The Journal of Machine Learning Research, 23(1), 5232-5270.

[10] Frankle, J., & Carbin, M. (2018). *The lottery ticket hypothesis: Finding sparse, trainable neural networks.* arXiv preprint arXiv:1803.03635.

[11] Ha, D., Dai, A., & Le, Q. V. (2017). *Hypernetworks.* arXiv preprint arXiv:1609.09106.

[12] He, X. O. (2024). *Mixture of a million experts.* arXiv preprint arXiv:2407.04153.

[13] Hennessy, J. L., & Patterson, D. A. (2017). *Computer architecture: A quantitative approach* (6th ed.). Morgan Kaufmann.

[14] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... & Amodei, D. (2020). *Scaling laws for neural language models.* arXiv preprint arXiv:2001.08361.

[15] Krajewski, J., Ludziejewski, J., Adamczewski, K., Pióro, M., Krutul, M., Antoniak, S., ... & Jaszczur, S. (2024). *Scaling laws for fine-grained mixture of experts.* arXiv preprint arXiv:2402.07871.

[16] Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., ... & Chen, Z. (2021). *Gshard: Scaling giant models with conditional computation and automatic sharding.* arXiv preprint arXiv:2006.16668.

[17] Liu, Z., et al. (2024). *DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models.* arXiv preprint arXiv:2401.06066.

[18] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). *Exploring the limits of transfer learning with a unified text-to-text transformer.* The Journal of Machine Learning Research, 21(1), 5485-5551.

[19] Rasley, J., Rajbhandari, S., Ruwase, O., & He, Y. (2020). *DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters.* In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 3505-3506).

[20] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). *Outrageously large neural networks: The sparsely-gated mixture-of-experts layer.* arXiv preprint arXiv:1701.06538.

[21] Sleator, D. D., & Tarjan, R. E. (1985). *Amortized efficiency of list update and paging rules.* Communications of the ACM, 28(2), 202-208.

[22] Tillet, P., Kung, H. T., & Cox, D. (2019). *Triton: an intermediate language and compiler for tiled neural network computations.* In Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (pp. 10-19).

[23] Zhao, S., Fang, Y., Liu, Z., Liu, Y., & Li, J. (2024). *HyperMoE: Towards Better Mixture of Experts via Transferring Among Experts*. ACL 2024. arXiv preprint arXiv:2402.12656.