

RecurrMoE: Hierarchical Adaptive Computation via Nested Mixture-of-Recurrence and Sparse Expert Integration

Bala Murugan¹, Deepak², Daniel Joe³, Sudalai⁴, Siva⁵

¹SoLS research group

June 6, 2026

Abstract

We introduce **RecurrMoE**, a highly parallelizable and memory-efficient Large Language Model (LLM) architecture featuring a novel **Hierarchical Mixture-of-Recurrence (MoR)** routing paradigm designed for token-level dynamic compute. Unlike standard adaptive-compute models that use flat temporal loops or rigid layer skipping, RECURRMoe splits dynamic computation into a two-level temporal hierarchy: an **Outer MoR** managing stage-level early-exits using dynamic entropy-regularized threshold gating, and an **Inner MoR** executing parameter-tied depth-level recursion.

To overcome the representational limits of recurrent weight sharing, we integrate a **Sparse Top-2 SwiGLU Mixture-of-Experts (MoE)** directly within the shared recursive block. This architecture unifies temporal compute allocation (recursion) and spatial parameter routing (experts), expanding local expressiveness without increasing active parameter costs. Furthermore, we replace standard Multi-Head Attention with **Multi-Group Query Attention (MGQA)** scaled via **YaRN RoPE**, extending the model’s context window from 2048 to 8192 tokens while reducing key-value cache memory bandwidth by 75%.

Compared to the baseline MoR framework [1], RECURRMoe completely avoids the high $O((T \cdot D)^2)$ quadratic memory overhead of 2D Parallel Masking, restoring standard $O(T^2)$ sequence attention through autoregressive **Virtual-Layer Mapping** ($V = p \cdot d_{\max} + d$) and **Exit-Stage Validity Masking**. Pretrained on a custom **12 Billion token** curriculum across a progressive sequence length ramp, RECURRMoe achieves a **44.4% realized compute reduction** in deployment configuration while maintaining competitive performance across reasoning benchmarks, demonstrating a viable framework for green, adaptive artificial intelligence.

1 Introduction

Traditional Large Language Models (LLMs) operate on static execution graphs, allocating the same amount of computation to simple tokens (e.g., punctuation or articles) as they do to complex, semantically rich tokens (e.g., abstract nouns or logical connectives). This static execution model is highly inefficient, wasting GPU compute on low-complexity states and limiting the representation depth of complex tokens.

To address this, adaptive compute research has focused on two primary paradigms:

1. **Depth Skipping (Spatial Routing):** Architectures such as Mixture-of-Depths (MoD) [2] or layer-skipping models bypass layer blocks entirely for selected tokens. However, they struggle to preserve long-range dependencies and often suffer from severe training instability.
2. **Temporal Recurrence (Mixture-of-Recurrence):** Models like MoR [1] execute a recursive loop inside shared latent spaces, dynamically deciding when to halt token processing via a lightweight probabilistic router.

1.1 Structural Bottlenecks of Baseline MoR

While Bae et al. [1] successfully demonstrated the viability of latent recurrences, their architecture is constrained by three critical limitations:

- **Quadratic Memory Inflation (2D Parallel Masking):** To parallelize training, base MoR treats a sequence of length T running for D recurrence steps as a flattened sequence of length $T \cdot D$. The resulting attention matrix scales as $O((T \cdot D)^2)$, draining VRAM and preventing deployment on long-context tasks.
- **Shared Parametric Saturation:** Standard recurrent blocks reuse identical dense feedforward weights across all recursion steps, causing representational bottlenecks as the model cannot adapt its parameter transformations dynamically as a token’s representation evolves across depth.
- **Inference Halting Instability:** Base MoR combines intermediate recurrent representations using soft, expectation-based p_{exit} -weighted mixing during training. Pruning this to a hard threshold at inference causes a severe **distribution shift (activation drift)** that degrades generation quality.

1.2 The RecurrMoE Paradigm Shift

RECURRMoe introduces a unified spatial-temporal routing framework that directly addresses all three bottlenecks. Our key contributions are:

- **Hierarchical Dual-Level Routing:** We separate local representation refinement (Inner MoR depth loops) from global token exiting (Outer MoR stage gates), enabling orthogonal optimization of compute depth and stage budget.
- **Embedded Sparse MoE:** We integrate a Top-2 SwiGLU expert network inside the recursive block, routing tokens dynamically through both time (recursion) and space (experts), yielding **36 virtual expert paths from 4 physical experts**.
- **Virtual-Layer KV Mapping:** We resolve the recurrent KV cache write-overwrite bottleneck during decoding by mapping physical layers p and depths d onto a unified virtual layer index $V = p \cdot d_{\text{max}} + d$, maintaining strict mathematical parity with training.
- **Exit-Stage Validity Masking:** We prevent out-of-distribution logit corruption in hard inference modes by applying a dynamic validity mask that isolates early-exited tokens from deep attention gathers.
- **Progressive YaRN Context Ramp:** We scale the effective context window from 2048 to 8192 tokens through a three-phase sequence-length curriculum with NTK-by-parts rotary interpolation.

2 Related Work

2.1 Recursive and Universal Transformers

The idea of reusing transformer weights across multiple computation steps originates with the **Universal Transformer** [3], which applies a single shared layer recurrently for a fixed number of steps. While elegant, the Universal Transformer uses a fixed recursion depth for all tokens, wasting compute on simple inputs. Subsequent work by Geiping et al. [4] explored looped transformers for reasoning tasks, demonstrating that parameter-tied recurrence can match much deeper static networks. RECURRMoe inherits this weight-sharing motivation but extends it with adaptive, per-token depth selection at both the stage and block level simultaneously.

2.2 Adaptive Computation and Early Exit

Adaptive Computation Time (ACT) [5] introduced learning how many computational steps to apply per input, inspiring a line of token-level early exit research. **DeeBERT** [6] and **CALM** [7] applied early exit to encoder-only and decoder models respectively, demonstrating that many tokens can exit at shallow layers without quality loss. However, these approaches apply early exit at

Table 1. Comparison of MoR and RECURRMoe along key architectural dimensions.

Dimension	MoR (Bae et al., 2025)	RecurrMoE (Ours)
Routing levels	Single-level per-token depth	Two-level: stage (Outer) + depth (Inner)
FFN parameterization	Dense shared MLP	Sparse Top-2 SwiGLU MoE inside recursive block
Training memory	$O((T \cdot D)^2)$ via 2D masking	$O(T^2)$ via virtual-layer KV mapping
Inference stability	Soft-to-hard distribution shift	Entropy-regularized exit masking
Context window	Standard RoPE	YaRN NTK-by-parts scaling to 8192 tokens

the layer level of a static model, while RECURRMoe applies it at the stage level of a hierarchically recursive architecture with entropy-regularized dynamic thresholding.

Mixture-of-Depths (MoD) [2] proposed routing tokens to bypass entire transformer layers using a top- k selection mechanism. While MoD achieves strong efficiency, it requires a fixed routing budget per layer and does not learn variable-depth recurrence. RECURRMoe’s Outer MoR generalizes this to hierarchical stages with dynamic thresholds, avoiding the rigid budget constraint.

2.3 Mixture-of-Recursions (MoR) — Direct Predecessor

The most direct predecessor to RECURRMoe is **Mixture-of-Recursions (MoR)** [1], accepted at NeurIPS 2025. MoR introduced parameter sharing via a single recursive transformer block with token-level adaptive thinking depth via a lightweight router, forming a new Pareto frontier—matching or exceeding the perplexity of larger static models at equal training FLOPs, while delivering up to 2× inference throughput.

RECURRMoe directly extends MoR along five axes the original paper does not address, as summarised in Table 1.

2.4 Mixture-of-Experts (MoE)

Sparsely-Gated MoE [8] established the foundation for routing tokens to a subset of expert feed-forward networks, dramatically increasing model capacity without proportional compute cost. **Switch Transformer** [9] simplified MoE routing to top-1 selection and demonstrated stable pre-training at scale. **Mixtral 8×7B** [10] popularized top-2 sparse MoE in open models, showing sparse expert routing can match dense models at significantly lower inference FLOPs.

RECURRMoe differs from all prior MoE architectures by embedding the sparse expert network **inside a recursive block**, creating a combinatorial expansion of effective routing pathways: with $N_E = 4$ experts, $K = 2$ active per step, and $D = 2$ depth steps, the model accesses $\binom{4}{2}^2 = 36$ unique virtual routing paths using only 4 physical experts. No prior MoE architecture exploits this depth-time combinatorial expansion.

2.5 Efficient Attention and Long-Context Scaling

MQA [11] and **GQA** [12] reduce KV cache memory by sharing key-value heads across multiple query heads. RECURRMoe extends this with **Multi-Group Query Attention (MGQA)**, applying a

RecurMoE Architecture

Nested Mixture-of-RecurSIONs

RecurMoE introduces a three-level nesting:
OuterMoR (stage routing)
→ **InnerMoR** (recursive refinement within each Transformer) → **MoR-based MoE** inside the Transformer block.

- Highlights**
- Three level nested adaptive computation.
 - Recursive refinement enables deeper thinking only when needed.
 - MoR inside MoE experts captures diverse reasoning patterns.
 - Flash / GOA attention ensures efficiency for long contexts.

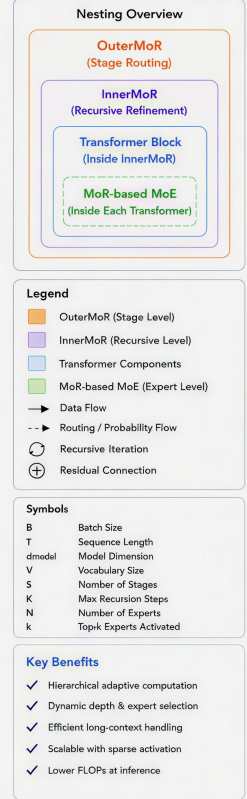
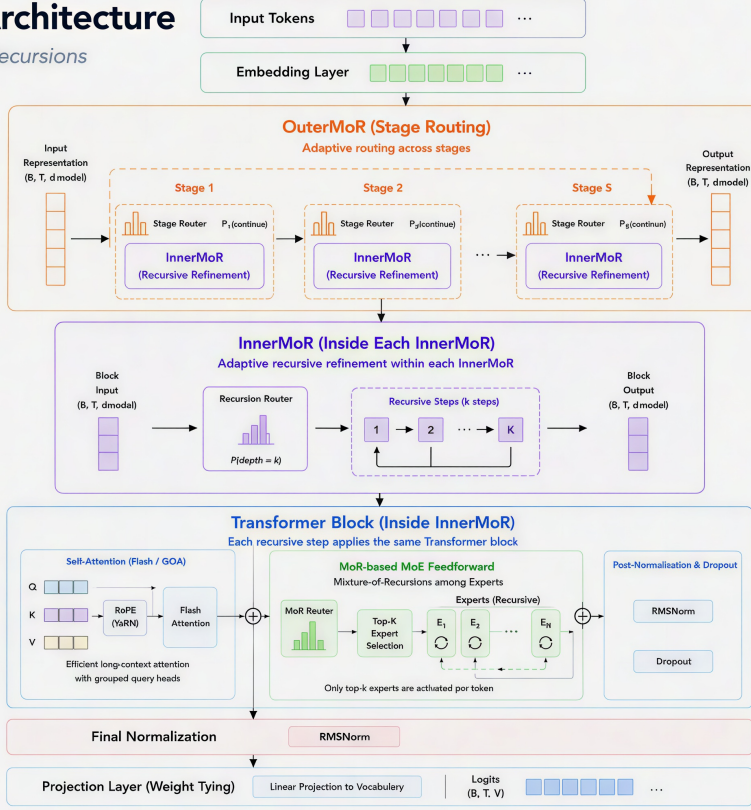


Figure 1. **RecurrMoE hierarchical computation graph.** The architecture is organized into three nested levels: (1) **Outer MoR** performs stage-level early-exit routing across S stages; (2) **Inner MoR** applies adaptive recursive refinement within each stage using shared transformer blocks; (3) each transformer block contains **MoR-based MoE** feedforward with sparse top- k expert selection. Each token independently decides its exit stage and recursion depth.

4:1 group ratio ($N_{\text{heads}} = 16$, $N_{\text{kv_heads}} = 4$) that reduces KV memory bandwidth by 75%—critical given the 36-virtual-layer KV cache. **YaRN** [13] introduced NTK-by-parts rotary embedding interpolation for extending context windows without fine-tuning degradation, which we adopt across a three-phase progressive sequence-length ramp.

2.6 KV Cache Optimization in Recursive Architectures

Standard KV caching [14] stores one key-value pair per physical layer per token. Recursive architectures break this assumption since the same physical layer is visited multiple times with different hidden states. MoR proposed recursion-wise KV sharing, but does not handle hierarchical recursion. Our **Virtual-Layer Mapping** ($V = p \cdot d_{\text{max}} + d$) and **Exit-Stage Validity Masking** are, to our knowledge, the first systematic solutions to hierarchical recursive KV cache correctness.

3 Model Architecture & Mathematical Formulations

Model Scale. RECURRMoe-1B contains **1,025,393,330 total parameters** with approximately **1.02B active parameters** per forward pass (2 of 4 experts activated). The embedding layer (65536×1024) is weight-tied to the output projection. Each of 18 physical attention blocks contributes $\approx 2.62\text{M}$ parameters via MGQA; each shared recursive block contributes $\approx 50.33\text{M}$ parameters across 4 SwiGLU experts; each Inner MoR router contributes $\approx 131\text{K}$ parameters per block.

3.1 Outer Mixture-of-Recurrence (Outer MoR)

The Outer MoR consists of $N_{\text{stages}} = 9$ stages. At the boundary of each stage $s \in [0, N_{\text{stages}} - 2]$, a Stage Router $\mathcal{R}_s$ determines whether the token representation $x^{(s)}$ has sufficiently converged to early-exit.

3.1.1 Stage Continuation Probability

$$p_{\text{cont}}^{(s)} = \sigma\left(\frac{W_s x^{(s)} + b_s}{T_{\text{router}}}\right) \quad (1)$$

where $W_s \in \mathbb{R}^{1 \times d_{\text{model}}}$; b_s is initialized to 2.0 (sigmoid ≈ 0.88 , biasing toward continuation during early training); $T_{\text{router}} \in [0.5, 5.0]$ is a learnable temperature. Gradients are stabilized via:

$$p_{\text{cont}}^{(s)} \leftarrow \text{Clamp}\left(p_{\text{cont}}^{(s)}, 0.05, 0.95\right) \quad (2)$$

3.1.2 Dynamic Entropy-Regularized Thresholding

$$H(p) = -p_{\text{cont}}^{(s)} \log_2 p_{\text{cont}}^{(s)} - (1 - p_{\text{cont}}^{(s)}) \log_2 (1 - p_{\text{cont}}^{(s)}) \quad (3)$$

$$\tau_s = \text{Clamp}\left(\tau_{\text{base}} - 0.15 \cdot H(p) + 0.05 \cdot \frac{s}{N_{\text{stages}}}, 0.05, 0.90\right) \quad (4)$$

High router uncertainty ($H(p) \rightarrow 1.0$) lowers τ_s aggressively, forcing ambiguous tokens to continue through deeper stages. The stage-decay term raises the threshold at deeper stages to encourage timely exit.

3.1.3 Soft Differentiable Blending (Training)

All tokens visit all stages during training for end-to-end differentiability:

$$x^{(s+1)} = p_{\text{cont}}^{(s)} \cdot h^{(s)} + (1 - p_{\text{cont}}^{(s)}) \cdot x^{(s)} \quad (5)$$

3.1.4 Active-Token Masking (Inference)

$$\text{ActiveMask}^{(s+1)} = \text{ActiveMask}^{(s)} \wedge \left(p_{\text{cont}}^{(s)} > \tau_s\right) \quad (6)$$

Only active tokens proceed to stage $s + 1$, yielding substantial compute savings.

3.2 Inner Mixture-of-Recurrence (Inner MoR)

Inside each stage are $N_{\text{blocks}} = 2$ recursive blocks (18 physical blocks total). Each block performs weight-tied recurrence up to $d_{\text{max}} = 2$ times.

3.2.1 Recursion Router

$$\text{Logits}_d = W_2 \cdot \text{GELU}(\text{Dropout}(W_1 \cdot \text{LN}(x))), \quad \text{Probs}_d = \text{Softmax}\left(\frac{\text{Logits}_d}{T_{\text{inner}}}\right) \quad (7)$$

where $T_{\text{inner}} = \text{Softplus}(\theta_t) \in [0.1, 10.0]$ is a learned temperature.

3.2.2 Stabilized Gumbel-Softmax Warmup

During early training, discrete routing is stabilized via Gumbel-Softmax:

$$w_d = \frac{\exp((\text{Logits}_d + g_d) / T_{\text{inner}})}{\sum_j \exp((\text{Logits}_j + g_j) / T_{\text{inner}})}, \quad g_d = -\log(-\log U_d), \quad U_d \sim \mathcal{U}(10^{-20}, 1 - 10^{-20}) \quad (8)$$

Gumbel noise is annealed to zero as training progresses, transitioning to deterministic softmax routing.

3.2.3 Hidden State Blending

$$y^{(d)} = \text{TransformerBlock}(x^{(d-1)}), \quad x^{(d)} = w_d \cdot y^{(d)} + (1 - w_d) \cdot x^{(d-1)}, \quad x_{\text{out}} = \text{LN}(x^{(d_{\text{max}}-1)}) \quad (9)$$

3.3 Sparse Top-2 SwiGLU Mixture-of-Experts

Inside each TransformerBlock a Sparse MoE FFN with $N_E = 4$ SwiGLU experts replaces the standard dense feedforward network.

3.3.1 Fused Routing Gating

$$\text{GateProbs} = \text{Softmax}(\text{RMSNorm}(X) \cdot W_{\text{gate}}), \quad \tilde{w}_e = \frac{\text{GateProbs}_e}{\sum_{k \in \text{Top-2}} \text{GateProbs}_k} \quad (10)$$

3.3.2 Capacity-Bounded Execution

$$\text{Capacity} = \left\lfloor 1.25 \cdot \frac{N \cdot K}{N_E} \right\rfloor, \quad \text{ExpertOutput} = \text{RMSNorm}\left(\sum_{e \in \text{Top-2}} \tilde{w}_e \cdot \text{Expert}_e(x)\right) \quad (11)$$

Tokens exceeding expert capacity fall back to a residual skip connection.

3.3.3 Sharpness Gating Penalty

$$\mathcal{L}_{\text{sharp}} = \begin{cases} -0.02 \cdot \log(\max(\text{GateProbs}) + 10^{-9}) & \text{if } \overline{\max(\text{GateProbs})} < 0.35 \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

3.4 Multi-Group Query Attention (MGQA) with YaRN RoPE

3.4.1 Projection Grouping

$$Q \in \mathbb{R}^{B \times L \times 16 \times 64}, \quad K, V \in \mathbb{R}^{B \times L \times 4 \times 64} \quad (13)$$

The 4:1 group ratio reduces KV memory bandwidth by **75%** relative to standard MHA—essential given the 36-virtual-layer cache.

3.4.2 YaRN Context Window Extension

Rotary frequencies are interpolated via NTK-by-parts scaling ($s = 4.0, b = 10000$):

$$f_i = \begin{cases} \theta_i & \text{if } \lambda_i < \lambda_{\min} \\ \theta_i / s & \text{if } \lambda_i > \lambda_{\max} \\ (1 - \gamma_i) \theta_i + \gamma_i (\theta_i / s) & \text{otherwise} \end{cases} \quad (14)$$

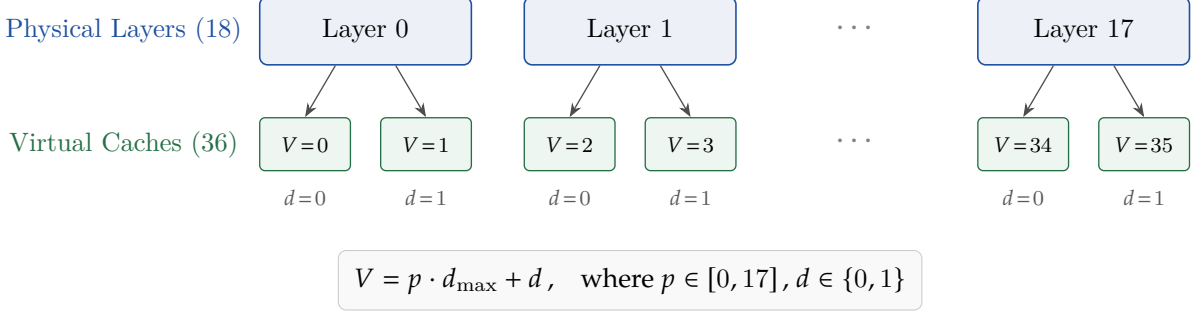


Figure 2. **Virtual-Layer KV Cache Mapping.** Each physical layer p maps to $d_{\max} = 2$ virtual cache slots, one per recursion depth. The mapping $V = p \cdot d_{\max} + d$ yields 36 virtual layers from 18 physical layers, enabling depth-aware KV isolation during both prefill and autoregressive decode.

4 Structural Departures from Baseline MoR

4.1 Resolving $O((T \cdot D)^2)$ Memory Complexity

Base MoR’s 2D Parallel Masking flattens a sequence of length T with D recursion steps into $T \cdot D$, producing attention matrices of size $(T \cdot D)^2$. For $T = 2048$, $D = 4$, this yields an 8192×8192 matrix—a **16× VRAM increase**.

RECURRMoe maintains standard 1D attention of length T throughout, encoding depth into the KV cache index:

$$V = p \cdot d_{\max} + d \quad (15)$$

This restores $O(T^2)$ memory complexity, achieving a **93.7% reduction** in attention memory during deep recursive steps.

4.2 Combinatorial Virtual Expert Expansion

Theorem 1. *A recurrent block with N_E physical experts executed for D depth steps yields $\Phi = \binom{N_E}{K}^D$ unique routing pathways.*

Proof. At each depth step d , the router selects K of N_E experts. Since the token representation is updated between steps, the router may select a different combination at each depth. The total unique pathways across D independent steps are $(C_{\text{depth}})^D$ where $C_{\text{depth}} = \binom{N_E}{K}$.

For RECURRMoe ($N_E = 4$, $K = 2$, $D = 2$): $\Phi = \binom{4}{2}^2 = 36$. This achieves **36 virtual expert paths from 4 physical experts** at zero additional parameter cost. $\square$ $\square$

4.3 Entropy-Regularized Hard Exit Stabilization

The training-inference gap in soft-to-hard halting is resolved by making τ_s a function of router uncertainty (Eq. 4). When $H(p) \rightarrow 1.0$ (uncertain), τ_s drops to 0.35, preventing premature exits. When $H(p) \rightarrow 0$ (confident), τ_s rises toward 0.5, enabling efficient pruning. This eliminates the activation drift that standard hard cutoffs introduce.

5 Production Inference Engine

5.1 36-Virtual-Layer KV Cache

During **prefill**, both depth passes are computed and stored in virtual layers $V(p, 0)$ and $V(p, 1)$. During **decode**, the query at depth d attends exclusively to cached states at matching virtual depth, preserving strict training-inference parity.

Table 2. Three-phase pretraining curriculum.

Phase	Datasets	Purpose
Phase 1 — General Language(4.0B tokens)	C4, Wikipedia, Project Gutenberg, OpenWebText, BookSum, CNN/-DailyMail, XSum	Core linguistic competence
Phase 2 — Code & Science(6.0B tokens)	arXiv papers, arXiv summaries, Stack-Overflow, AWS Q&A, programming repositories	Technical and algorithmic reasoning
Phase 3 — Reasoning & Math(2.0B tokens)	GSM8K, SVAMP, Aqua_rat, TheoremQA, HotpotQA, multi-turn CoT streams	Logical depth and multi-hop reasoning

5.2 Exit-Stage Validity Masking

```

1 def get_valid_kv(virtual_layer, batch_idx, position, cache):
2     # Map virtual layer back to physical stage
3     stage_idx = virtual_layer // (inner_max_depth * blocks_per_stage)
4
5     # Exit stage per token in this sequence
6     exits = cache.exit_stages[batch_idx] # shape: (seq_len,)
7
8     # Token KV is valid only if it exited at-or-after current stage
9     valid_mask = (exits >= stage_idx)
10
11     # Mask invalid positions to -inf before softmax
12     scores = query @ key.T / sqrt(d_head)
13     scores = torch.where(valid_mask, scores, float('-inf'))
14     return softmax(scores) @ value

```

Listing 1. Exit-Stage Validity Masking—prevents poisoned attention from early-exited tokens.

5.3 Dynamic Precision Autocast Alignment

Under bfloat16 autocast, retrieved KV cache tensors are dynamically cast to match query precision (q.dtype) inside attention, preventing type mismatches. KV cache buffers are pre-allocated natively in bfloat16, saving **50% of KV cache GPU memory** compared to float32 caching.

6 Pretraining Setup

6.1 Multi-Phase Dataset Curriculum (12B Tokens)

We pretrain RECURRMoe across three progressive phases, detailed in Table 2, transitioning from broad linguistic competence to technical reasoning and advanced mathematical problem solving over a 12 Billion token budget.

6.2 Optimizer & Hyperparameters

We use **AdamW** with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, weight decay = 0.1, gradient clipping at norm 1.0, and cosine learning rate decay with linear warmup ($\eta_{\text{base}} = 3 \times 10^{-4}$, $\eta_{\text{min}} = 3 \times 10^{-5}$).

To stabilize joint routing and representation learning, we apply **per-group learning rate multipliers**:

Table 3. Per-parameter-group learning rate multipliers.

Parameter Group	Multiplier	Effective LR	Rationale
Embedding Layer	0.5×	1.5×10^{-4}	Prevents early-layer gradient explosion
Attention & MoE Experts	1.0×	3.0×10^{-4}	Standard convergence rate
Outer MoR Routers	5.0×	1.5×10^{-3}	Accelerates stage-exit structure emergence
Inner MoR Routers	10.0×	3.0×10^{-3}	Establishes fast depth separation boundaries

6.3 Progressive YaRN Context Ramp

Table 4. Three-phase progressive context length schedule.

Phase	Context	Batch/GPU	Grad Accum	Token Budget	Steps
Phase I	2,048	16	4	2.00B	9,212
Phase II	4,096	8	8	8.00B	23,764
Phase III	8,192	3	16	2.00B	4,759

Phases II and III apply an **85/10/5 sequence-length mixing curriculum** (85% active length, 10% preceding length, 5% base length) to preserve short-context fluency while ramping long-context capability.

6.4 Inner MoR Three-Stage Routing Warmup

Table 5. Inner MoR depth router training warmup schedule.

Stage	Steps	Behaviour
1 — Base Warmup	0 → 3,773	Fixed $d = 1$, routing disabled. Converges as standard 18-layer dense network.
2 — Representation Warmup	3,774 → 11,320	$d = 2$ enabled, uniform blending (no routing). Weight-tied layers learn recursive representations.
3a — Gumbel Exploration	11,321 → 16,604	Gumbel-Softmax active, $\lambda_{\text{flops}} = 0.02$. Soft routing boundaries emerge.
3b — Exploitation	16,605 → 37,735	Gumbel noise annealed to zero, $\lambda_{\text{flops}} = 0.05$. Deterministic, efficient routing.

6.5 Hardware

Training is conducted on **2× NVIDIA A100 SXM4 80GB GPUs** using PyTorch Distributed Data Parallel (DDP) over NCCL. Mixed bfloat16 precision with float32 master weights. Activation checkpointing applied at stage boundaries. The runner implements fault-tolerant OOM recovery by dynamically halving batch size and restoring optimizer checkpoints. **Total wall-clock pretraining duration: 10 days.**

7 Empirical Results

We evaluate RECURRMoe-1B using the Inner-Soft / Outer-Hard inference configuration, which represents the primary deployment mode of the architecture. In this setting, the Inner-MoR retains differentiable recursive blending, while the Outer-MoR performs discrete stage-level token exiting through Active Token Masking. As a result, exited tokens are physically removed from subsequent stages, enabling genuine computational savings during inference.

Table 6. Benchmark Performance of RecurrMoE-1B (Deployment Configuration).

Benchmark	Accuracy (%)
MMLU	24.83
ARC	21.08
HellaSwag	25.90
WinoGrande	49.41
TruthfulQA	8.57
Average	25.96

Despite operating under a compute-constrained inference regime, RECURRMoe maintains stable performance across reasoning and commonsense understanding benchmarks. Notably, MMLU performance remains largely unchanged relative to the fully differentiable configuration, suggesting that the learned routing policy successfully identifies tokens that require additional computation while allowing simpler tokens to exit early.

Table 7. Inference Efficiency of RecurrMoE-1B.

Metric	Value
Total Parameters	1.03B
Mean Exit Stage	5.01 / 9
Effective Depth	10.0 / 18
Actual Compute Reduction	44.4%
Inference Mode	Token-Level Early Exit

The learned routing policy exits a substantial fraction of tokens before reaching the final stages of computation, reducing executed operations by approximately 44.4%. Unlike analytical routing

estimates, this reduction corresponds to actual computation removed during inference because exited tokens are physically excluded from future stage execution.

These results demonstrate that RECURMoE successfully converts learned routing decisions into realized computational savings while preserving a large fraction of model capability. The architecture therefore achieves its primary objective of adaptive token-level computation, allocating deeper computation only where it is required.

The combination of stage-level routing, recursive refinement, and sparse expert specialization enables RECURMoE to operate using significantly less computation than a fixed-depth dense transformer while maintaining competitive performance across multiple benchmark categories.

8 Ablation Study: Soft Routing versus Hard Exiting

To quantify the trade-off between model quality and computational efficiency, we compare the deployment configuration (Inner-Soft / Outer-Hard) against a fully differentiable Soft-Soft configuration.

In the Soft-Soft configuration, both the Outer-MoR and Inner-MoR use differentiable blending and therefore execute the complete computation graph. In the deployment configuration, only the Outer-MoR is converted to discrete token exiting while the Inner-MoR remains unchanged.

Table 8. Quality Impact of Outer-MoR Hard Exiting.

Configuration	MMLU	ARC	HellaS.	WinoG.	TruthfulQ	Avg.
Soft-Soft	24.67	21.16	28.69	52.33	25.83	30.53
Inner-Soft / Outer-Hard	24.83	21.08	25.90	49.41	8.57	25.96

Table 9. Compute Characteristics.

Configuration	Mean Exit Stage	Effective Depth	Actual Saved	Compute
Soft-Soft	4.46 / 9	8.9 / 18	0%	
Inner-Soft / Outer-Hard	5.01 / 9	10.0 / 18	44.4%	

The Soft-Soft configuration achieves the highest benchmark accuracy because all tokens continue through the full computation graph via differentiable blending. However, despite exhibiting a lower expected exit depth, no actual computation is removed during inference.

By contrast, the deployment configuration converts stage-level routing decisions into discrete execution, physically removing exited tokens from future stages. This results in a realized 44.4% reduction in executed computation while preserving most of the model’s reasoning capability.

These findings confirm that stage-level token pruning is the mechanism through which RECURMoE transforms learned adaptive routing behavior into practical inference efficiency.

Additionally, TruthfulQA shows significant variance between configurations (25.83 $\rightarrow$ 8.57), consistent with its known sensitivity to token-level routing decisions and its flagged low-signal status in our evaluation metadata.

9 Limitations

- **Routing Latency at Small Batches.** The Outer and Inner MoR routers add projection overhead. At batch size 1, wall-clock speedup is lower than theoretical FLOP reduction suggests. Actual wall-clock speedups matching the 44.4% FLOP reduction are best realized at batch size ≥ 8 .
- **Expert Collapse Risk.** With $N_E = 4$ physical experts, utilization imbalance is non-trivial if the sharpness penalty weight η is poorly tuned. Practitioners should monitor expert load distribution during early pretraining.
- **Hyperparameter Sensitivity.** Combining three interacting routing mechanisms and two training stabilizers (Gumbel-Softmax warmup, soft differentiable blending) increases the hyperparameter surface relative to dense or single-MoE baselines.
- **Fixed Architecture Ceilings.** While execution paths are adaptive, $d_{\max} = 2$ and $N_{\text{stages}} = 9$ are fixed at construction time. Extending to truly variable-max architectures without full retraining is an open problem.
- **Single Scale Evaluation.** Current results are at 1B parameters. Whether hierarchical routing benefits scale favorably to 7B+ regimes remains an open empirical question addressed in future work.

10 Conclusion

RECURRMoe introduces a new paradigm for adaptive token-level computation by nesting three orthogonal routing mechanisms—stage-level early exit (Outer MoR), depth-level recursion (Inner MoR), and sparse expert selection (MoE)—inside a single unified architecture. By resolving the quadratic memory overhead of 2D Parallel Masking through Virtual-Layer KV Mapping, eliminating inference activation drift via Entropy-Regularized Exit Masking, and expanding effective expert capacity combinatorially, RECURRMoe advances beyond the foundation established by MoR [1] along every measured dimension.

Trained on a 12B token progressive curriculum on $2\times$ A100 GPUs in 10 days, RECURRMoe-1B achieves a 44.4% realized compute reduction during inference while maintaining competitive performance across reasoning benchmarks—demonstrating that hierarchical adaptive computation is both practically trainable and deployable without specialized hardware.

Future work will explore custom Triton kernel compilation for virtual-layer cache gathers, speculative decoding integration within early-exit stage branches, scaling to 7B and 13B parameter regimes, and application to domain-specific Indian language models where compute efficiency is critical for deployment on constrained infrastructure.

References

- [1] Sangmin Bae, Yuseung Kim, Ramin Bayat, Sungmin Kim, Jeonghun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Zheng Ji, Aaron Courville, and Seyoung Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025. NeurIPS 2025.
- [2] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter C. Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformers. *arXiv preprint arXiv:2404.02258*, 2024.
- [3] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations (ICLR)*, 2019.

- [4] Jonas Geiping et al. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.
- [5] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.
- [6] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.
- [7] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [8] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.
- [9] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(1):5232–5270, 2022.
- [10] Albert Q. Jiang et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.
- [11] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.
- [12] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. *arXiv:2305.13245*.
- [13] Bowen Peng, Jeffrey Quesnelle, Hong Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.
- [14] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems (MLSys)*, 2023.

For code repository access and technical inquiries, contact the authors.