Dynamic Mixture of Experts: An Auto-Tuning Approach for Efficient Transformer Models

Published at ICLR 2025

Original Authors: Y. Guo et al.
Presented by: Fengnan Li

Duke B&B

Nov, 2025

Outline

1. Introduction & Motivation
2. Methodology: Top-Any Gating
3. Methodology: Adaptive Architecture
4. Methodology: Loss Function
5. Experiments & Results
6. Conclusion

The Bottleneck of Large Models

The Trend:

- Transformers have scaled massively (Billions $\rightarrow$ Trillions of parameters).
- Dense scaling is computationally expensive (quadratic/linear cost increase).

The Solution: Sparse Mixture of Experts (MoE)

- Replaces dense Feed-Forward Networks (FFN) with multiple "Experts".
- **Conditional Computation:** Only activates a subset of parameters per token.
- **Benefit:** High Capacity (Total Params) with Low Compute (Active Params).

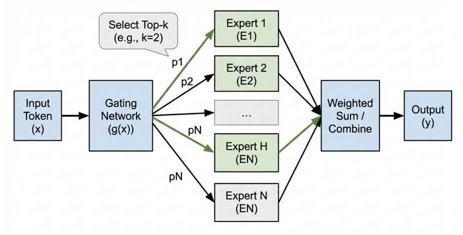


Diagram of Standard Sparse MoE Architecture

The Problem: Hyperparameter Sensitivity

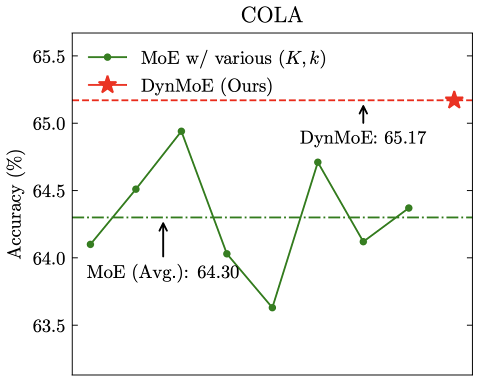
While MoE is efficient, training it is difficult due to sensitive hyperparameters.

Critical Hyperparameters

- K **(Total Experts):** How many experts should we have? (8, 16, 64?)
- k **(Top-k Activation):** How many experts should process each token? (1, 2, 4?)

The Consequences:

- Performance fluctuates significantly (1%-3%) based on the choice of (K, k).
- Finding the optimal pair requires expensive **Grid Search** or Ablation Studies.
- Different tasks require different settings (e.g., QNLI needs $K = 16$, RTE needs $K = 8$).



(a) Performance Fluctuation Illustration

The Proposal: Dynamic MoE (DynMoE)

Core Question: *Can we develop a training strategy that* **automatically determines** *both the number of experts and the activation count during training?*

DynMoE Framework Overview:

1. **Top-Any Gating:** A mechanism that allows tokens to autonomous decide how many experts to use (0 to N).
2. **Adaptive Training Process:** A lifecycle management system that adds new experts when needed and removes idle ones.
3. **Auxiliary Loss:** A specialized loss to ensure the model remains sparse and numerically stable.

1. Top-Any Gating Mechanism

Standard MoE uses **Softmax** (Competition). DynMoE uses **Sigmoid** (Independence).

Mechanism Details: Scoring & Discrete Decision

1. **Cosine Similarity Scoring:** Calculate $s(x)$ between input token x and expert weights W_g. Each expert has a corresponding column in W_g.

$$s(x) = \frac{\langle x, W_g \rangle}{||x|| \cdot ||W_g||}$$

2. **Discrete Gating Decision** $g(x)$**:** Apply the **Sign Function** to convert soft scores into binary activations $\{0, 1\}$. This compares the sigmoid-scaled similarity score $\sigma(s(x))$ against a learnable per-expert threshold $\sigma(G)$.

$$g(x) = \text{sign}(\sigma(s(x)) - \sigma(G))$$

- This gating allows a token to activate **any number** of experts (from 0 to N).

1. Top-Any Gating Mechanism

Continuing from Part 1: After determining active experts using $g(x)$.

Mechanism Details: Expert Output Aggregation

3. **Expert Output Aggregation:** Compute the simple average of outputs from all activated experts. Here, $k = \sum g(x)$ is the count of active experts for the current token.

$$y = \frac{1}{k} \sum_{g(x)_e > 0} E_e(x)$$

Crucial Design Questions:

- **Scale Mismatch:** Why use simple averaging $(1/k)$ instead of weighted sum (like standard MoE)? We'll discuss this soon.
- **Gradient Flow:** We used a non-differentiable 'sign' function in $g(x)$. How do we pass gradients to learn W_g and G? (Next Slide!)

Passing Gradients via Straight-Through Estimator

The Problem: The derivative of the ‘sign‘ function used in the previous step is 0 almost everywhere. Gradients cannot flow back.

The Solution (STE Trick): We use the **Straight-Through Estimator** in the computational graph to bypass the sign function during backpropagation.
Let $\text{soft} = \sigma(s(x)) - \sigma(G)$.

Computational Graph Implementation

$$g_{\text{final}}(x) = (g(x) - \text{soft}).\text{detach}() + \text{soft}$$

- **Forward Pass:** Terms cancel out. The result is exactly the discrete $g(x)$.
- **Backward Pass:** ‘.detach()‘ blocks gradient flow in the first term.

$$\frac{\partial \mathcal{L}}{\partial g_{\text{final}}} \approx \frac{\partial \mathcal{L}}{\partial \text{soft}}$$

Gradients successfully flow back to update weights and thresholds.

Output Aggregation: Why Simple Average?

Once experts are selected, how do we combine their outputs?

Equation (6): Simple Averaging

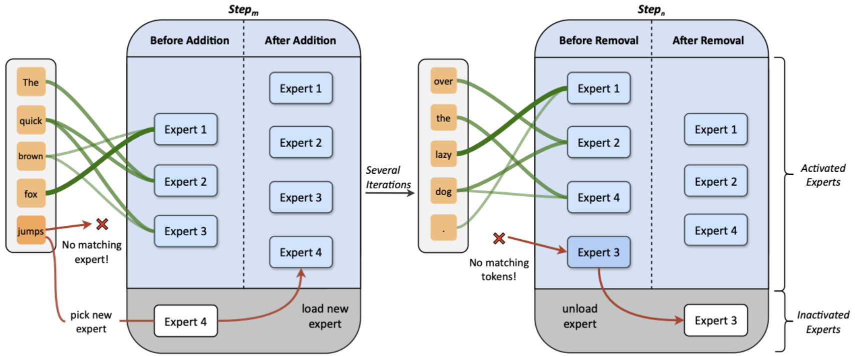
Let $k = \sum g(x)$ be the number of activated experts.

$$y = \frac{1}{k} \sum_{g(x)_e > 0} E_e(x)$$

Remark 3.1: The "Scale Mismatch" Problem

- Why not use scores as weights (like standard MoE)?
- In our "Independent Gating" approach, experts learn different thresholds.
- **Example:** Expert A might be activated with score 0.15 (Low Threshold), while Expert B is activated with 0.9 (High Threshold).
- Using scores as weights would unfairly penalize Expert A, even though both are valid experts.
- **Conclusion:** We ignore magnitude and treat all active experts equally.

2. Adaptive Training Process (Overview)



- **Timing:** The system checks and acts every **100-300 iterations** to stabilize the process.
- **Growth (Adding Experts):** If tokens are "homeless" ($R_S \neq 0$), a new expert is **added** and initialized using the R_S direction.
- **Pruning (Removing Experts):** If an expert is completely idle ($R_E = 0$), it is **removed** to maintain sparsity and efficiency.

Phase 1: Recording "Homeless" Tokens

The Problem of $k = 0$

Since thresholds are independent, it is possible that a token activates **zero** experts. This means the current experts are insufficient to cover this token's semantics.

The Accumulator (R_S):

- During the monitoring window, we identify tokens where $\text{sum}(g(x)) = 0$.
- We accumulate their embedding vectors into a buffer called R_S:

$$R_S = R_S + \sum_{i=1}^{N} \mathbb{1}_{k_i=0} \cdot x_i$$

- R_S effectively represents the "average direction" of the unsolved problems.

Phase 2: Adding New Experts

If $R_S \neq 0$ (i.e., there are homeless tokens), we recruit a new expert.

Initialization Strategy (Crucial)

We create Expert $K+1$. How do we initialize it?

$$W_{g,K+1} = \frac{R_S}{||R_S||}, \quad G_{K+1} = 0$$

- **Weight Initialization:** We set the new expert's vector to point exactly in the direction of the homeless tokens (R_S). This guarantees they will match next time.
- **Threshold Initialization:** We set $G = 0$ (Sigmoid output 0.5). This is a "neutral" starting point, ensuring the expert is easily activated immediately.

Adaptive Process: Removing Experts (Pruning)

The Adaptive Training Process also prunes underutilized experts to maintain efficiency.

Mechanism: Pruning Idle Experts

- **Routing Record (R_E):** During the monitoring window, the system records the activation count for every expert.
- **Condition:** We check if there is an expert e such that its recorded activation time is zero ($R_E^e = 0$).
- **Action:** If the expert e has been completely idle, it is **removed** from the model structure to free up computational resources.

3. Sparse and Simple Gating Loss

Without constraints, Top-Any Gating might activate ALL experts to minimize loss. We need a regularization term.

$$\mathcal{L}_{aux} = \underbrace{||W_g^T W_g - I_K||_2}_{\text{Diversity Loss}} + \underbrace{\frac{1}{K}\sum_{e=1}^{K} ||w_{g,e}||_2}_{\text{Simplicity Loss}}$$

- **Diversity Loss ($O(W^4)$):**
  - Forces $W_g^T W_g \approx I$.
  - This pushes expert vectors to be **Orthogonal**.
  - If vectors are orthogonal, a single token cannot be similar to all of them simultaneously $\rightarrow$ **Enforces Sparsity**.
- **Simplicity Loss ($O(W^2)$):**
  - Penalizes the norm of weight vectors (L2 Regularization).
  - **Why needed?** Diversity loss (4th order) can cause exploding gradients. Simplicity loss provides a linear counter-force to stabilize values and prevent Sigmoid saturation.

Experiment Overview: 4 Key Questions

The authors designed experiments to answer four specific questions:

Performance

Q1: Can DynMoE achieve competitive performance vs. fixed settings?
Q2: Can it handle different modalities (Vision/Language)?

Efficiency & Insight

Q3: Does it maintain sparsity (Efficiency)?
Q4: What insights does it offer for MoE design?

Q1: Competitive Performance (GLUE Benchmark)

Baseline: Standard MoE with various grid-searched (K, k).

Observations:

- Green Lines (Standard MoE): Show high variance. Optimal K, k changes for every task.
- Red Star (DynMoE): Consistently performs above average or at the top.
- **Top-2 Accuracy Times:** DynMoE achieves the highest score (3.00), proving it is the most robust setting across tasks.

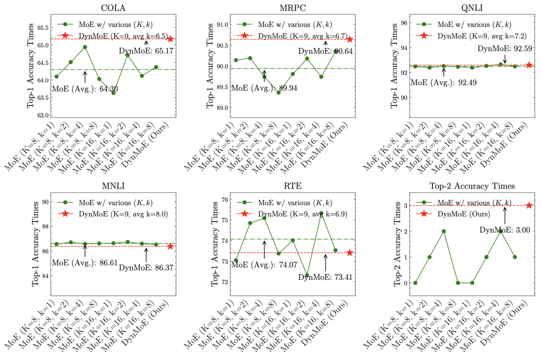


Figure 4: GLUE Results (Fluctuation vs. Robustness)

Q2: Modalities & Scales

DynMoE was tested on Vision (DomainBed) and Vision-Language (LLaVA) tasks.

Task / Backbone	Baseline (MoE-LLaVA)	DynMoE (Ours)
Vision (DomainBed)	72.8 (avg)	72.4 (avg)
VL (StableLM-1.6B)	76.7	**77.4**
VL (Qwen-1.8B)	76.2	**76.4**
VL (Phi-2-2.7B)	77.6	**77.9**

Table: Performance Comparison on Vision-Language Tasks (VQA)

Conclusion: DynMoE consistently matches or outperforms carefully tuned baselines across different model scales and modalities.

Q3: Sparsity & Efficiency (Does Auto-Tuning explode?)

"Does Auto-Tuning lead to parameter explosion?" No.

Inference Efficiency

DynMoE automatically learns to activate **fewer** parameters than the fixed baseline ($k = 2$).

- **StableLM:** Activated **1.75B** (DynMoE) vs 2.06B (Baseline).
  $\rightarrow \sim$**15% reduction in inference compute.**
- **Qwen:** 2.19B (DynMoE) vs 2.24B (Baseline).

Training Efficiency

- Training FLOPs are comparable to Standard MoE.
- **Major Gain:** Saves the massive compute cost of "Ablation Studies" (running the training 8+ times to find the best K, k).

Q4: Insights for Architecture Design

The DYNMOE framework provided two key insights that can guide the future design of MoE models.

Insight 1: Layer-wise Sparsity

- **Bottom Layers (Shallow):** Tokens tend to activate experts more uniformly (higher k), reflecting the need for broader feature extraction.
- **Top Layers (Deep):** Tokens select fewer experts, often just one expert, as representations converge for final tasks.
- **Implication:** MoE structure may only be required for **bottom layers**. Top layers could remain dense, saving computation without performance loss.

Insight 2: Emergence of Shared Experts

- **Observation:** Visualizing the learned thresholds (G) showed that one expert per layer consistently learns a significantly **lower threshold**.
- **Pattern:** This low threshold makes that expert more readily activated by almost all tokens, acting as a "Generalist".
- **Validation:** This self-emergent behavior is consistent with the manual design choice of incorporating shared experts in models like DeepSeek-MoE.

Summary

1. **Problem:** DynMoE addresses the inefficiency of hyperparameter tuning in MoE training.
2. **Method:**
   - **Top-Any Gating** (Sigmoid + STE) allows flexible activation.
   - **Adaptive Process** adds/removes experts based on real-time demand.
3. **Result:**
   - Competitive performance with State-of-the-Art.
   - Reduced inference costs (15% less params).
   - Removed the need for grid search.

Code available at: `https://github.com/LINs-lab/DynMoE`

Thank You!

Questions?