# Beyond Top-k: A Comparative Study for Dynamic Routing Mechanisms in Mixture of Experts Models

Tarunyaa Sivakumar, Steven Su, Areeb Alam
*University of Pennsylvania School of Engineering and Applied Science*
*CIS 6200 Advanced Topics in Deep Learning*

***Abstract*—Mixture of Experts (MoE) models offer a scalable and compute-efficient alternative to dense transformer architectures by activating only a small, relevant subset of expert subnetworks per input token. However, the effectiveness of these models depends critically on the routing strategy used to determine which experts are activated. In this paper, we present a comprehensive comparative study of expert routing mechanisms within MoE-augmented GPT-2 models. Specifically, we evaluate fixed Top-$k$ routing alongside three dynamic strategies: Top-$p$, Top-Any, and Entropy-Adaptive routing.**

**Our experiments, conducted on the SlimPajama dataset, assess these routing policies across multiple metrics including cross-entropy loss, perplexity, inference latency, memory usage, FLOPs, and expert utilization. Results show that dynamic strategies like Top-$p$ and Top-Any significantly outperform the dense baseline and the fixed Top-$k$ routing in terms of both accuracy and expert load balancing, while offering compute trade-offs.**

**Through visualizations and correlation analyses, we further investigate the interaction between input complexity and expert selection behavior. Our findings highlight the importance of adaptive routing in achieving efficient, expressive, and well-utilized MoE models, offering actionable insights for the design of next-generation sparse neural architectures.**

## I. Introduction

### A. Mixture of Experts (MoE)

Mixture of Experts (MoE) models are a class of neural networks that enhance the scalability of deep learning by activating only a subset of specialized subnetworks—called experts—during inference. Instead of computing the full forward pass across all parameters, MoE models dynamically select a few relevant experts per input token. This selective execution drastically reduces inference and compute costs. MoE architectures are particularly effective in tasks like language modeling, where the complexity of inputs can vary widely across tokens.

### B. Expert Routing

The core mechanism enabling MoE models is expert routing, which determines which experts are activated for a given input token. An effective routing mechanism selects experts such that model performance remains comparable to dense execution, but with significantly lower computational overhead. Thus, routing plays a critical role in balancing model efficiency, accuracy, and expert specialization.

### C. Conventional Routing Mechanisms

A widely used routing approach in early MoE models is **Top-K** routing. Here, a gating network computes relevance scores between each token and expert, then selects the top-$k$ scoring experts. Only these are activated, and their outputs are aggregated—typically via a weighted sum informed by the gating probabilities.

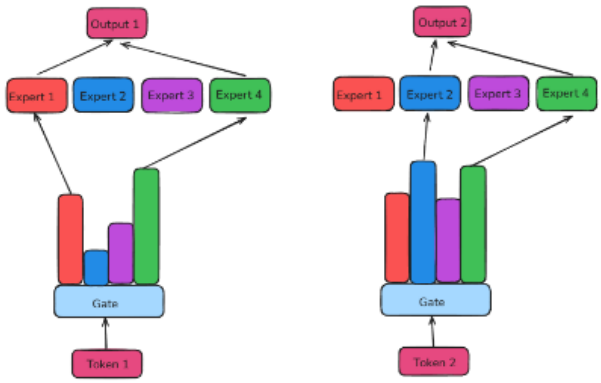



Fig. 1. Top-K routing mechanism: selects a fixed number of top-scoring experts.

While simple and effective, Top-K suffers from rigidity. Regardless of the token's difficulty, it always activates the same number of experts. This can lead to inefficient use of compute—overprovisioning for easy inputs and underutilization for hard ones. Additionally, Top-K often leads to poor load balancing: some experts are overused while others remain idle, which can harm training convergence and expert specialization.

### D. Dynamic Routing Mechanisms

Dynamic routing addresses the limitations of fixed Top-K policies by allowing the number of activated experts to vary per token. These mechanisms adapt expert usage based on the token's complexity or confidence scores, improving both computational efficiency and model flexibility.

**Top-P routing** selects the minimal set of experts whose cumulative gating scores exceed a predefined threshold $p$. This allows token-specific variability in expert count, adapting compute effort to the model's confidence.

**Top-Any routing**, proposed in this work, introduces per-expert trainable thresholds combined with sigmoid gating. Each expert is activated if its sigmoid-transformed score exceeds its threshold. This allows flexible, fine-grained expert selection per token, enabling models to achieve high expressiveness with adaptive compute.

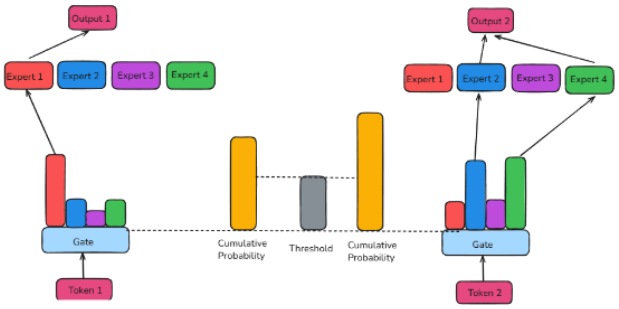



Fig. 2. Top-P routing: activates the smallest set of experts whose cumulative gating scores exceed a threshold $p$.

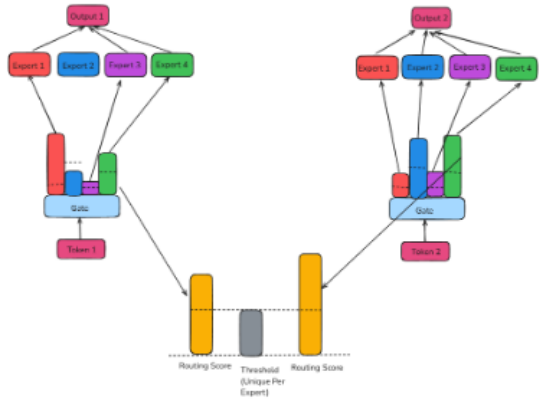



Fig. 3. Top-Any routing: uses per-expert trainable thresholds and sigmoid gating for token-level activation.

### E. Motivation

As reasoning-based models and AI agents become more prominent, compute demands are projected to grow exponentially. Jensen Huang, CEO of NVIDIA, recently predicted a 100× increase in compute demand, largely fueled by the rise of these increasingly capable yet compute-intensive systems. Traditional dense models activate all parameters for every token, leading to significant inefficiencies—particularly during inference. Mixture of Experts (MoE) architectures offer a compelling alternative by selectively activating only a small subset of specialized subnetworks, or experts, per token. For instance, DeepSeek-V3 contains 671 billion parameters in total, but activates only 37 billion per token—less than 6% of the model—dramatically reducing inference cost. Motivated by this, our work investigates how routing strategies in MoE models affect inference-time compute. By intelligently routing tokens—based on their difficulty or uncertainty—we aim to further reduce compute costs while maintaining, or even enhancing, model performance. This approach has the potential to make large-scale models significantly more efficient and practical for deployment.

## II. Methods

In this section, we present the architecture of our baseline (dense) model and our MoE backbone. We also present the implementation of chosen routing strategies. Finally, we present the auxiliary loss and the training specifications.

### A. GPT-2 Vanilla [1]

We adopt pretrained GPT-2 as our baseline architecture due to its structural simplicity and well-documented robust performance across a variety of language tasks.

GPT-2 is one of the earliest large-scale Transformer-based language models. Released by OpenAI in 2019, it was trained on WebText, a curated dataset of approximately 8 million high-quality web pages. GPT-2 was designed for the objective of causal language modeling, predicting the next token given previous context, and demonstrated strong zero-shot generalization across downstream tasks without task-specific fine-tuning.

Architecturally, GPT-2 consists of a stack of identical Transformer decoder blocks, each composed of masked multi-head self-attention, layer normalization, and feed-forward layers. Its simplicity and scalability made it a cornerstone model in the evolution of large language models.

OpenAI released GPT-2 in four sizes, varying in parameter count and model depth. The specifications are summarized in Table I.

To construct our dense, baseline model, we build on pretrained GPT-2 Large by adding 57 identical decoder blocks to the architecture. The additional decoder blocks are architectureally identical to all existing decoder blocks and are initialized with the same weights and biases as the last decoder block of the original pretrained GPT-2 Medium model. The final dense, baseline model has roughly 1.15 billion parameters. Note that the additional decoder blocks are required to scale up the baseline model's size to roughly equal the size of the MoE model explained in the next section to enable fair comparisons. We refer this baseline, dense model as GPT-2 Vanilla in subsequent sections. After initializing the model, we further train GPT-2 Vanilla till convergence before experimenting.

### B. GPT-2 MoE

In this section, we describe the procedure through which we constructed GPT-2 MoE, the MoE architecture based on GPT-2 Medium that we use in this study. To describe the GPT-2 MoE architecture, we present the implementation of our 1) *Expert Network* 2) *Router Network* 3) *MoE Layer* and 4) *GPT-2 MoE Decoder Block*.

**1) Expert Network**. The Expert Network in our implementation consists of a linear layer that up-projects input dimension of 1024 to 4096, the GELU activation function, and a linear layer that down-projects the dimension of the output from 4096 back to 1024, sequentially.

Each Expert Network's weights and biases are initialized with the exact same weights and biases of the original GPT-2 Vanilla's Feed Forward Layer scaled down by the number of experts per MoE Layer, which is 9 in our implementation.

| Model | Parameters | Hidden Size | Decoder Blocks | Attention Heads | Expert Layers | Experts Per Layer |
|---|---|---|---|---|---|---|
| GPT-2 Small | 117M | 768 | 12 | 12 | x | x |
| GPT-2 Medium | 345M | 1024 | 24 | 16 | x | x |
| GPT-2 Large | 762M | 1280 | 36 | 20 | x | x |
| GPT-2 XL | 1.5B | 1600 | 48 | 25 | x | x |
| GPT-2 Vanilla | 1.15B | 1024 | 87 | 16 | x | x |
| GPT-2 MoE | 1.15B | 1024 | 24 | 16 | 12 | 9 |

TABLE I
SPECIFICATIONS OF GPT-2 MODEL VARIANTS RELEASED BY OPENAI, AS WELL AS OUR CUSTOM GPT-2 VANILLA AND GPT-2 MOE IMPLEMENTATION

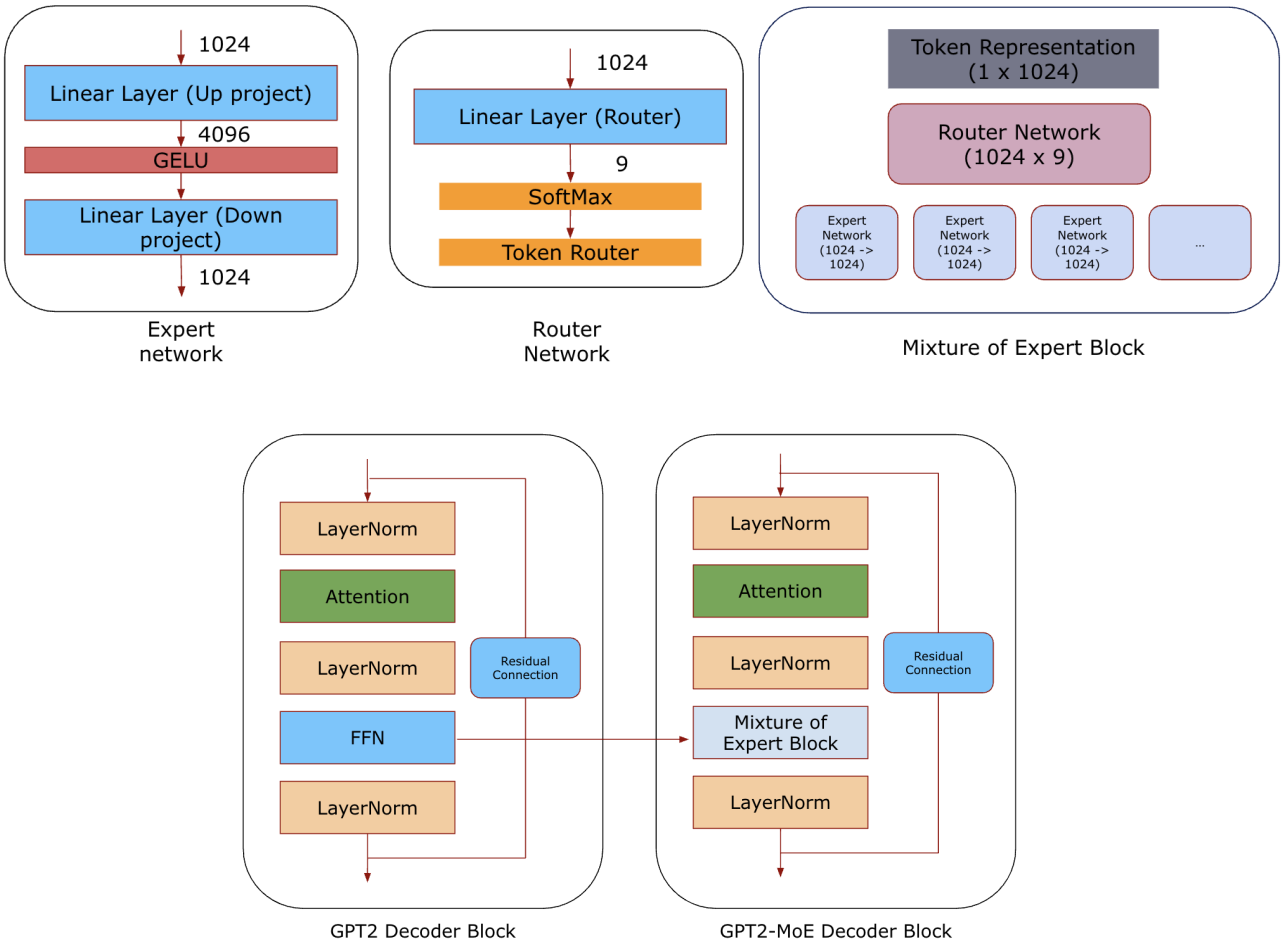



Fig. 4. ARCHITECTURE OF EACH GPT-2 MOE COMPONENT

**2) Router Network**. In our implementation, the Router Network is a single linear layer that converts input size 1024 to an output size of 9, which assigns a logit to each Expert Network in a MoE Layer.

Then, the logits go through SoftMax and are converted into probabilities assigned to each Expert Network. Finally, custom routing strategies described in subsequent sections are responsible for routing tokens to individual Expert Networks based on their assigned probabilities.

The weights and biases of the Router Network is initialized via Gaussian noise.

**3) MoE Layer**. A MoE Layer is designed to replace the Feed Forward Layers in GPT-2's decoder blocks. It consists of an Router Network that route tokens to 9 architecturally identical Expert Networks.

**4) GPT-2 MoE Decoder Block**. We replace the Feed Forward Layer in a GPT-2 Decoder Block with a MoE Layer to derive a GPT-2 MoE Decoder Block. Then, we convert a decoder block to a MoE Decoder Block every 2 decoder blocks in GPT-2 Medium to derive GPT2-MoE used for experiments in this model. The final parameter count of GPT-2 MoE is around 1.15 billion.

The architecture of each components in GPT-2 MoE is illustrated in Figure 4.

### C. Top-k Routing [2]

Top-$k$ gating activates a fixed number $k$ of experts per token. Let $x \in \mathbb{R}^d$ be the token representation, and $W_g \in \mathbb{R}^{d \times E}$ be the gating network's parameters. The gating logits are computed as:

$$g(x) = \text{softmax}(W_g^\top x)$$

Then, the top-$k$ experts with highest gating scores are selected:

$$\text{Top-}k(x) = \arg\overset{(k)}{\max}_{i \in [E]} g(x)_i$$

The MoE output is the weighted combination of selected expert outputs:

$$y = \frac{1}{\sum_{e \in \text{Top-}k(g(x))} g(x)_e} \sum_{e \in \text{Top-}k(g(x))} g(x)_e \cdot \text{Expert}_e(x)$$

Despite its success, Top-$k$ requires careful tuning of $k$ and forces all tokens to activate the same number of experts regardless of complexity or task specificity.

### D. Top-p Routing [3]

Top-$p$ routing is a dynamic expert selection mechanism that activates a variable number of experts per token based on the model's confidence. It addresses a key limitation of Top-$k$ routing: the assumption that each token should always use a fixed number of experts, regardless of complexity.

Let $g(x) \in \mathbb{R}^E$ denote the softmax-normalized gating scores over $E$ experts for a given token $x$. These scores reflect the model's belief in each expert's suitability to handle the token.

To perform Top-$p$ routing:

- First, the gating scores $g(x)$ are sorted in descending order.
- Then, we compute the smallest subset $S \subseteq [E]$ of expert indices such that the cumulative gating mass satisfies:

$$\sum_{i \in S} g(x)_i \geq p$$

- This set $S$ defines the dynamically chosen experts for token $x$. The number of experts $|S|$ varies per token and depends on the shape of the gating distribution.

This adaptivity gives Top-$p$ the abilitycan to allocate more compute to complex tokens and less to simple ones—yielding both flexibility and potential compute savings.

Formally, let $P_i$ denote the $i$th highest gating score and $I = (i_1, i_2, \ldots, i_E)$ the sorted expert indices. Then the number of activated experts $t$ is:

$$t = \min \left\{ k \in \{1, \ldots, E\} : \sum_{j=1}^{k} P_j \geq p \right\}$$

The final gating mask is:

$$g_i^*(x) = \begin{cases} g_i(x), & \text{if } i \in S \\ 0, & \text{otherwise} \end{cases}$$

where $S = \{i_1, i_2, \ldots, i_t\}$.

### E. Top-Any Routing [2]

Top-Any gating addresses two limitations of Top-$k$: (1) the need for hyperparameter tuning, and (2) the fixed expert count across tokens. Inspired by multi-label classification, this method treats each expert as an independent binary decision.

First, cosine similarity is computed between input token $x$ and expert embeddings $W_g \in \mathbb{R}^{d \times E}$:

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

A sigmoid is applied to squash scores into $(0, 1)$:

$$\tilde{g}(x) = \sigma(s(x))$$

Experts are activated if their sigmoid score exceeds a learned per-expert threshold $G \in \mathbb{R}^E$:

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

The number of activated experts for token $x$ is:

$$k = \sum g(x)$$

The final output is a uniform average over active experts:

$$y = \frac{1}{k} \sum_{e : g(x)_e > 0} \text{Expert}_e(x)$$

To prevent the issue of token dropping (i.e., no expert selected), a fallback rule at inference time is introduced:

$$\tilde{g}(x)_i = \begin{cases} \sigma(s(x)_i), & \text{if } i = \arg\max_j \sigma(s(x)_j) \text{ and } \sum g(x) = 0 \\ 0, & \text{otherwise} \end{cases}$$

### F. Entropy-Adaptive Routing

We implement an entropy-adaptive routing mechanism that dynamically adjusts the number of experts activated per token based on the router's confidence.

Inspired by entropy-based gating regularization [4], which penalizes high-entropy gating distributions in the loss function to encourage more decisive expert selection, our approach instead *uses entropy as a control signal* for routing behavior. Specifically, for each token, we compute the entropy of its routing distribution and normalize it to the range $[0, 1]$ to decouple it from the number of experts. This normalized entropy value is then linearly scaled to a target number of experts $k \in [\texttt{min\_k}, \texttt{max\_k}]$, with lower entropy (i.e., higher confidence) activating fewer experts and higher entropy activating more. In our setup with 4 total experts, we set $\texttt{min\_k} = 1$ and $\texttt{max\_k} = 9$, enabling the model to allocate compute adaptively based on token-level uncertainty. This routing strategy balances computational efficiency with robustness by concentrating expert use where it is most needed.

### G. Auxiliary Loss

To encourage balanced expert utilization and stable routing behavior, we adopt an auxiliary loss composed of two components: a *load balancing loss* and a *router Z-loss*. These terms are applied during training to regularize the expert selection process and stabilize the router logits.

Let the following notation be defined:

- $B$: batch size
- $T$: sequence length
- $N = B \times T$: total number of tokens in the batch
- $k$: number of top experts selected per token
- $E$: total number of experts
- $C_i$: number of times expert $i$ is selected in the batch
- $f_i = \frac{C_i}{k \cdot N}$: usage fraction of expert $i$

**Load Balancing Loss** is defined as:

$$\mathcal{L}_{\text{balance}} = E \cdot \sum_{i=1}^{E} f_i^2 \tag{1}$$

This is the scaled squared $L_2$ norm of the expert usage fractions. The loss is minimized when all experts are used equally, i.e., $f_i = \frac{1}{E}$ for all $i$.

**Router Z-loss** penalizes large magnitudes of the router logits $z_{bt} \in \mathbb{R}^E$ (pre-softmax scores for expert selection) for token at batch index $b$, time step $t$:

$$\mathcal{L}_z = \frac{1}{N} \sum_{b=1}^{B} \sum_{t=1}^{T} \|z_{bt}\|_2^2 \tag{2}$$

This term stabilizes routing decisions by minimizing the mean squared $L_2$ norm of the logits across all tokens.

**Total Auxiliary Loss** combines the two components with weights $\lambda$ and $\epsilon$:

$$\mathcal{L}_{\text{aux}} = \lambda \cdot (\mathcal{L}_{\text{balance}} + \epsilon \cdot \mathcal{L}_z) \tag{3}$$

In our experiments, we use $\lambda = 0.01$ and $\epsilon = 0.001$.

### H. Training Setup

We train our model using the standard causal language modeling objective on the `SlimPajama-627B` dataset, a deduplicated and cleaned version of the original *Pile* dataset curated for language model pretraining. `SlimPajama-627B` consists of high-quality English text sourced from a diverse range of domains including books, web pages, scientific articles, and open-source code. It is designed to retain the coverage and diversity of the original Pile while significantly reducing noise and redundancy through aggressive filtering and deduplication. The dataset is well-suited for evaluating the generalization and robustness of large language models in both academic and real-world settings.

Specifically, we use 10,000 samples from the training split and 1,000 samples from the evaluation split. The training loss is the sum of the token-level cross-entropy loss and the auxiliary loss described in the previous subsection.

We adopt an effective batch size of 32 and optimize using the Adam optimizer with a learning rate of $5 \times 10^{-5}$, scheduled linearly over the course of training. The models, including GPT-2 Vanilla and multiple variations of GPT-2 MoE with distinct routing strategies are trained until convergence using a single NVIDIA A100 GPU from Google Collab.

### I. Training Convergence Test

Tensorboard was used to ensure that a model was trained to convergence and to accordingly select the number of epochs and learning rate. Below is an example of the figures that we used to inform our choices. Evaluation loss (top left of

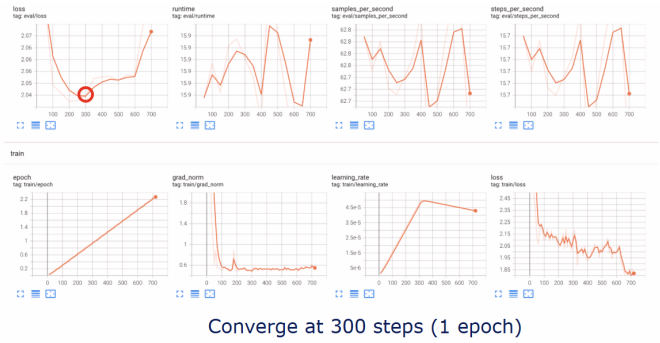



Fig. 5. Training and evaluation curves for a representative GPT-2 MoE model.

Figure 12) reaches its minimum at approximately 300 training steps, indicating convergence after a single epoch. Other subplots show runtime performance, throughput (samples and steps per second), learning rate schedule, gradient norm decay, and loss progression. These diagnostics collectively confirm stable optimization dynamics and effective early convergence.

## III. Experiments and Results

To evaluate the model, we construct the test set from the `SlimPajama-627B` dataset as described in the previous section. We stream 1,000 text examples from the test split using the Hugging Face `datasets` library. Each example is passed through a GPT-2-compatible tokenizer which converts raw text into token ID sequences. Tokenized sequences are concatenated and grouped into fixed-length segments of 1,024 tokens, matching the model's maximum context length. This chunking ensures uniform input shapes during evaluation and avoids padding artifacts. The resulting token chunks are structured as both `input_ids` and `labels` for autoregressive language modeling, where the model learns to predict each token given its preceding context. The processed dataset is then wrapped in a PyTorch `DataLoader` with a batch size of 4 to enable efficient batched evaluation.

All evaluations were conducted on an `AWS g5.12xlarge` instance. This instance is equipped with four NVIDIA A10G GPUs, which each features 24 GB of GPU memory. Evaluations were conducted on the finetuned models that were trained and uploaded to Hugging Face.

### A. Model Evaluation

#### 1. Loss

TABLE II
AVERAGE CROSS-ENTROPY LOSS PER MODEL.

| Model | Average Loss |
|---|---|
| Baseline | 9.4053 |
| Top-k | 8.3511 |
| Top-p | 8.3247 |
| Top-any | **8.3137** |
| Entropy Adaptive | 8.3216 |

TABLE III
MODEL PERPLEXITY ON THE TEST SET. LOWER PERPLEXITY INDICATES BETTER LANGUAGE MODELING PERFORMANCE.

| Model | Perplexity |
|---|---|
| Baseline | 12152.92 |
| Top-k | 4234.88 |
| Top-p | 4124.32 |
| Top-any | **4079.35** |
| Entropy Adaptive | 4111.80 |

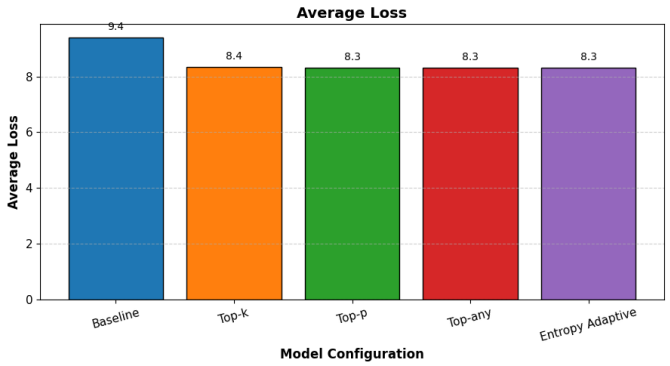



Fig. 6. Comparison of average cross-entropy loss across different expert routing strategies. Lower values indicate better predictive performance.

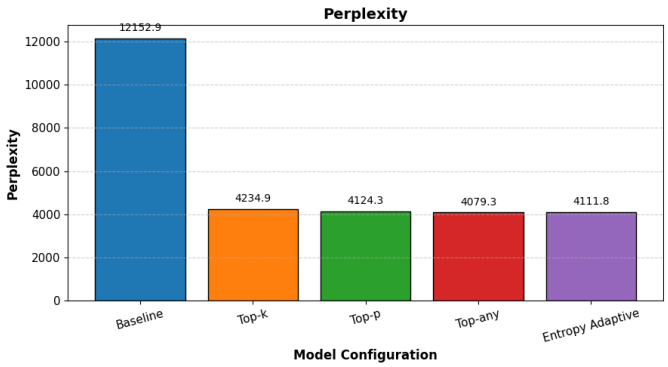



Fig. 7. Perplexity of each model on the SlimPajama-627B test set.

These results are directly computed as the mean token-level loss across the entire test dataset, with total loss accumulated as `total_loss+ = loss.item() × batch_tokens`, where `loss.item()` represents the scalar loss value (cross-entropy) returned by the model per token, and `batch_tokens` is the number of tokens in that batch.

All MoE configurations substantially outperform the baseline dense model, which records a loss of 9.41. The best-performing model, `Top-any`, achieves a 11.6% reduction in loss (8.31), while `Top-k`, `Top-p`, and `Entropy Adaptive` follow closely with approximately 11.2% to 11.5% improvements over the baseline. The consistent gain across all MoE variants suggests that sparsely-activated expert subnetworks improve model expressiveness and generalization without requiring full parameter activation.

The consistent performance gain across all MoE variants reflects the increased expressiveness of the model—defined here as the ability to capture and model more diverse patterns in the input space. MoE architectures enhance expressiveness by enabling expert specialization: each expert sub-network can focus on different linguistic phenomena, styles, or domains, thereby improving generalization without globally overfitting. As tokens are routed through tailored subnetworks, the model captures a richer diversity of behaviors, reducing the overall loss. This is a finding that has been validated by prior literature. In fact, in Shazeer et al. [5], their MoE model achieves significantly lower loss while using fewer computational resources in comparison to LSTM models.

## 2. Perplexity

Perplexity, defined as the exponential of the average loss, quantifies model uncertainty in next-token prediction. A lower perplexity indicates that the model assigns higher confidence to correct predictions.

The baseline model performs poorly with a perplexity of 12,153, indicating severe uncertainty in next-token prediction. All MoE configurations reduce perplexity drastically, with `Top-any` again achieving the lowest at 4,079—a 66.4% reduction. `Top-p` and `Entropy Adaptive` closely follow. These results confirm that expert routing enhances the model's confidence and sharpness in its output distribution.

## 3. Average Inference Time per Token

From Table IV, `Top-k` is the fastest configuration, requiring only 0.0718 ms per token. This represents a 42.4% improvement in latency compared to the baseline's 0.1246 ms. By activating a fixed number of experts, `Top-k` minimizes routing overhead.

In contrast, `Top-p`, `Top-any`, and `Entropy Adaptive` involve dynamic expert selection, which might incur additional overheads. These overheads can be computational in nature such as the calculations involving top-p sampling and entropy although we aren't quite sure.

TABLE IV
AVERAGE INFERENCE TIME PER TOKEN (IN MILLISECONDS). LOWER VALUES INDICATE FASTER EVALUATION.

| Model | Time per Token (ms) |
|---|---|
| Baseline | 0.1246 |
| Top-k | **0.0718** |
| Top-p | 0.1054 |
| Top-any | 0.1083 |
| Entropy Adaptive | 0.1263 |

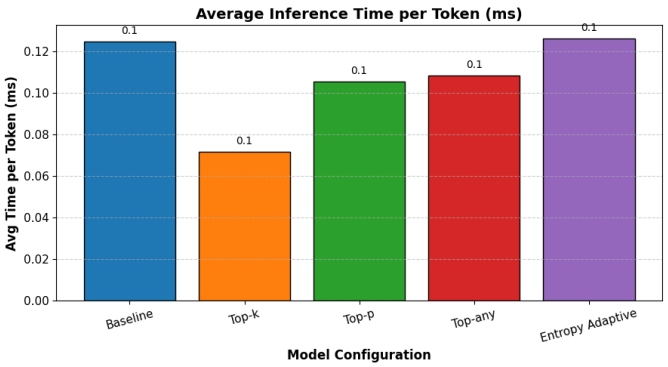



Fig. 8. Inference latency per token (in milliseconds) for each model configuration.

## 4. Average Memory Used per Batch

TABLE V
AVERAGE GPU MEMORY USED PER EVALUATION BATCH (IN MB). LOWER IS MORE MEMORY EFFICIENT.

| Model | Memory (MB) |
|---|---|
| Baseline | **2215.70** |
| Top-k | 4867.34 |
| Top-p | 5385.34 |
| Top-any | 5321.90 |
| Entropy Adaptive | 5318.37 |

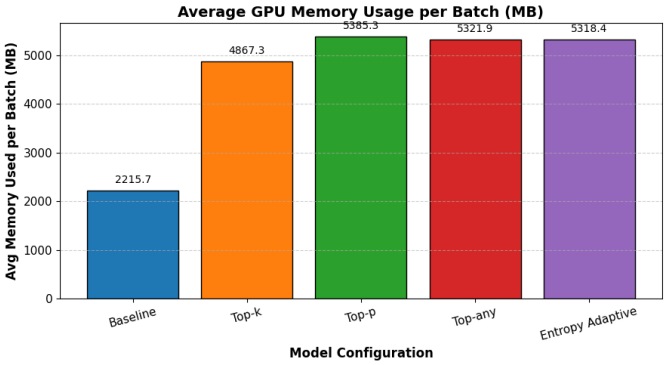



Fig. 9. Average GPU memory consumption per batch during evaluation.

Memory usage is computed as the mean of memory snapshots taken immediately before and after each forward pass.

Table V indicates that the baseline model is most memory-efficient at 2215.7 MB. MoE models generally consume 2.4x more memory on average due to expert parallelism. `Top-k` remains relatively efficient (4867.3 MB), while `Top-p`, `Top-any`, and `Entropy Adaptive` exceed 5300 MB. The elevated memory usage might be attributable to simultaneous expert execution and intermediate buffer overheads during routing. However, we aren't completely sure of why this is the case and this is a future research direction.

TABLE VI
TOTAL FLOATING-POINT OPERATIONS DURING EVALUATION (IN GFLOPS). LOWER IS MORE COMPUTE-EFFICIENT.

| Model | Total FLOPs (GFLOPs) |
|---|---|
| Baseline | 2,234,605.05 |
| Top-k | **787,077.66** |
| Top-p | 1,226,984.75 |
| Top-any | 1,108,924.11 |
| Entropy Adaptive | 983,194.32 |

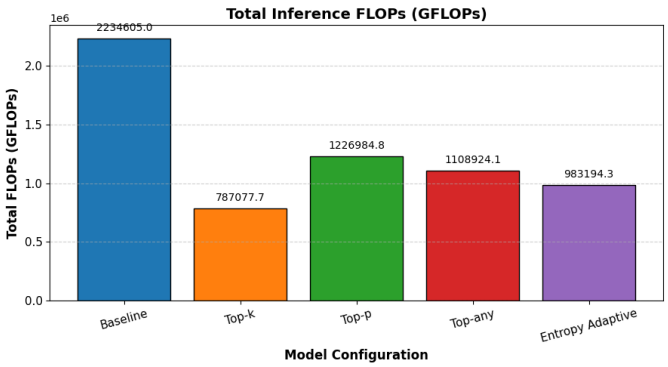



Fig. 10. Total number of floating-point operations performed during evaluation, measured in GFLOPs.

## 5. Total FLOPs

We use the PyTorch Profiler with `with_flops=True` to calculate FLOPs (Floating Point Operations) per batch. These are accumulated over the test set and normalized by token count to yield.

According to Table VI, MoE variants significantly reduce the total floating-point operations required during inference. `Top-k` is the most efficient at 787 GFLOPs, representing a 64.8% reduction compared to the baseline (2,234 GFLOPs). The reduction stems from sparsity: only a subset of the parameters is active per token. `Top-any` and `Top-p` consume moderately more compute due to more selection of experts, which increases the number of parameters evaluated per token.

## 6. Average Experts Activated per Token

TABLE VII
AVERAGE NUMBER OF EXPERTS ACTIVATED PER TOKEN. ENTROPY ADAPTIVE WAS NOT INSTRUMENTED FOR USAGE.

| Model | Avg Experts/Token |
|---|---|
| Baseline | 0.00 |
| Top-k | 2.00 |
| Top-p | **6.49** |
| Top-any | 5.28 |
| Entropy Adaptive | — |

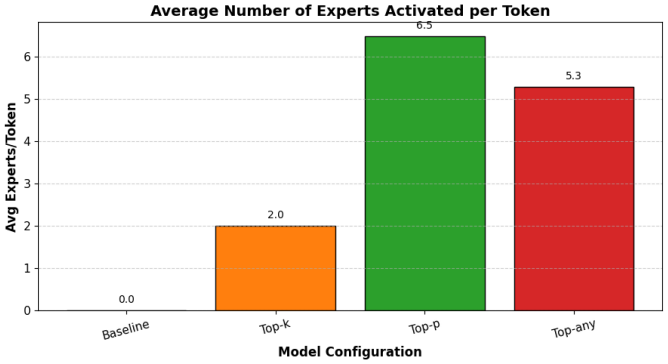



Fig. 11. Average number of experts activated per token across layers. Entropy Adaptive results were omitted due to computational errors during Evaluation.

Finally, Table VII outlines the average number of experts activated per token. `Top-k` maintains a fixed value of 2 experts, which enables predictable inference-time resource allocation. `Top-p` uses the most experts per token (6.49), yielding higher model expressiveness and accuracy, but at the cost of greater memory and compute. `Top-any` strikes a balance with 5.28 experts/token.

### 7. Layer-level Expert Usage metrics

We collected data on layer-wise average of expert activations, giving per-layer sparsity statistics. We implemented this to understand how the model utilizes its experts. For each input, the number of tokens routed to each expert in each layer is recorded via the model's internal usage counter buffers. After each forward pass, these counters are reset. We compute the average number of experts activated per token, normalized by the number of experts (e.g., 9 in this case), as well as the standard deviation of this count across all batches.

Below we have graphs that depict expert usage across transformer layers and expert IDs. Each point denotes the activation of a specific expert within a particular layer, and the size of the circle is proportional to the number of tokens routed to that expert during inference. Larger circles indicate heavier usage, reflecting the model's preference for certain experts.

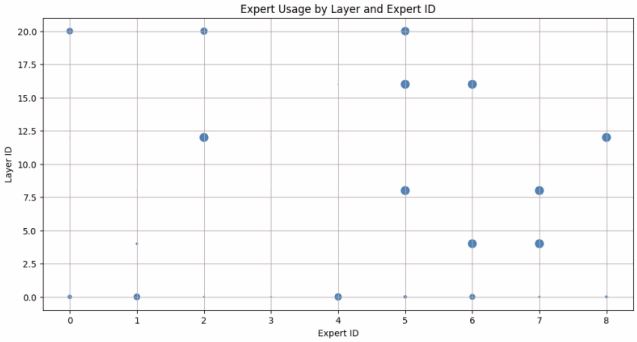



Fig. 12. Expert Usage by Layer and Expert ID under top-k routing policy.

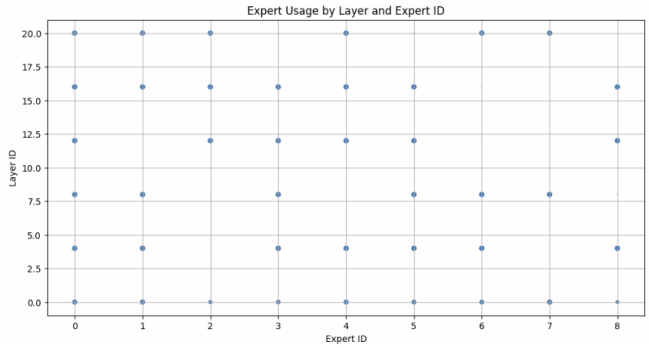



Fig. 13. Expert Usage by Layer and Expert ID under top-p routing policy.

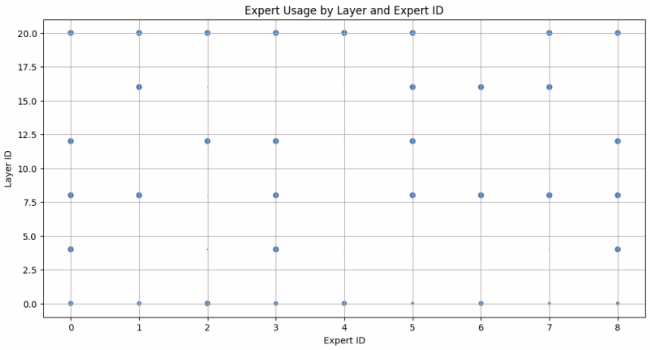



Fig. 14. Expert Usage by Layer and Expert ID under top-any routing policy.

In Figure 12, the `Top-k` policy exhibits a highly concentrated usage pattern. A few experts per layer dominate token routing, resulting in large circle markers for only a subset of experts. This indicates a lack of diversity in expert usage.

In contrast, Figure 13 shows that `Top-p` routing achieves a near-uniform distribution of token routing across all experts and layers. Every expert receives consistent number of tokens, suggesting successful load balancing. This is the most balanced configuration in terms of expert usage.

Figure 14 presents the `Top-any` configuration, which displays a broader activation pattern than `Top-k` but less uniformity than `Top-p`. A wide range of experts is used, though with some variation in frequency. This might indicate a practical compromise between expressiveness and computational efficiency.

The size of each circle in these plots reflects the number of tokens routed to that expert in the corresponding layer, providing an intuitive visualization of load distribution. These results reinforce the trade-off between rigidity of routing policu and load balancing. Fixed strategies like `Top-k` have limited expert diversity, while adaptive strategies like `Top-p` and `Top-any` more fully engages the model's experts.

### 8. Correlation of Expert Usage with Input Complexity

The input prompt complexity for each test example was calculated as the mean entropy across the model's output

probabilities.

Below we have scatter plots showing the relationship between input difficulty and the number of experts activated per token. Each point corresponds to a single evaluation example. The low correlation suggests that the model's routing behavior is relatively invariant to input complexity while a high correlation suggests that the model's routing behaviour is sensitive to the input complexity.

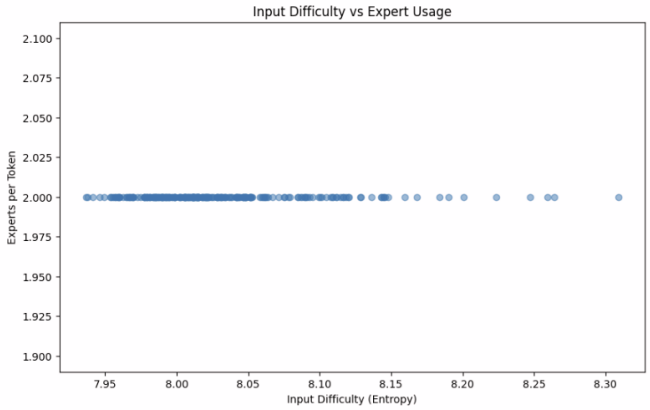



Fig. 15. Input prompt complexity against expert activated per token under top-k routing policy.

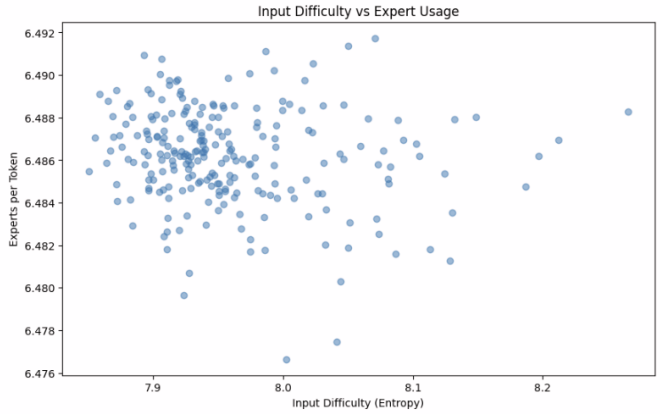



Fig. 16. Input prompt complexity against expert activated per token under top-p routing policy.

TABLE VIII
PEARSON CORRELATION BETWEEN INPUT DIFFICULTY AND EXPERT ACTIVATION ACROSS ROUTING POLICIES AND NUMBER OF EXPERTS.

| Routing Policy | Number of Experts | Correlation |
|---|---|---|
| Top-k | 9 | NaN |
| Top-p | 9 | -0.1474 |
| Top-any | 9 | -0.0398 |
| Top-k | 4 | NaN |
| Top-p | 4 | **0.4863** |

We analyze the correlation between input prompt entropy and the number of experts activated per token across different routing policies and configurations. Entropy serves as a proxy for input difficulty—the higher the entropy, the more uncertain the model is in generating a next-token prediction.

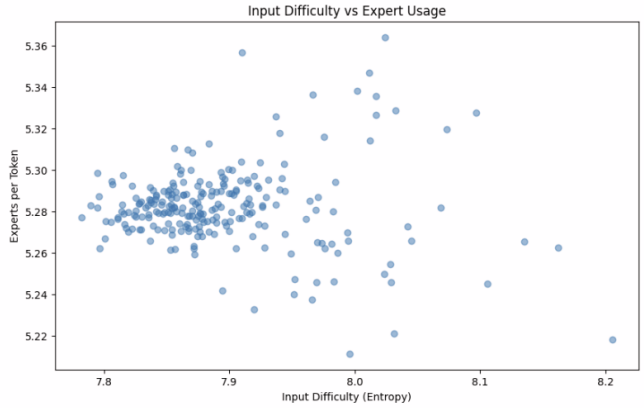
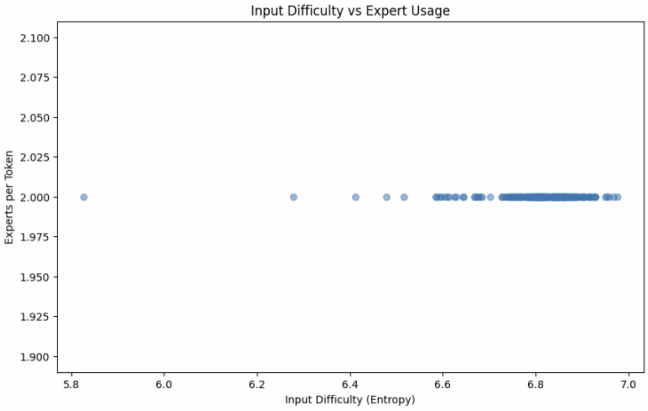
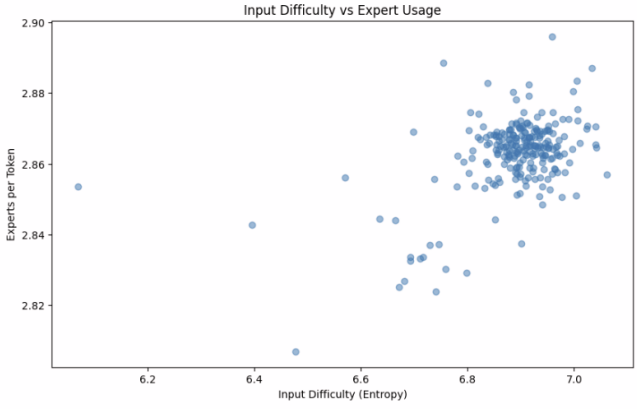



Fig. 17. Input prompt complexity against expert activated per token under top-any routing policy.




Under the Top-k policy, the number of experts per token is fixed. This results in a flat distribution across all samples, as clearly seen in Figures 15 and 18, where the expert count remains constant regardless of entropy. Consequently, the correlation is undefined (NaN), as there is no variance in the dependent variable. This reflects the rigidity of fixed routing, which cannot adapt based on input complexity.



The Top-p policy, by contrast, dynamically selects experts based on a probabilistic threshold. With 8 total experts (Figure 16), the observed correlation is weakly negative (–0.1474), suggesting a slight tendency to activate fewer experts on more complex inputs. This counterintuitive behavior could arise because a high input prompt entropy leads to the softmax distribution becoming more uniform. Under the Top-p routing strategy, experts are selected until the cumulative probability surpasses a predefined threshold p. When the softmax distribution is flatter because of the higher entropy, the cumulative probability accumulates more slowly, potentially leading to the selection of fewer experts to meet the threshold. This phenomenon has been discussed in recent studies [3].

Interestingly, when the number of experts is reduced to 4 (Figure 19), the correlation under Top-p becomes strongly positive (0.4863). This implies that with fewer routing choices, the model is able to dynamically activate more experts for a more complex input which is what we'd expect. This can be because fewer experts lead to an increase in the likelihood of overlapping routing probabilities which leads to broader activation to meet the top-p mass cutoff. This is an area of further research.

Top-any routing exhibits a near-zero correlation (–0.0398), indicating that expert count per token remains relatively similar regardless of input difficulty. We aren't sure why this is the case as we'd expect there to be fewer experts activated when the input prompt is simple and the router is given the choice to choose any number of experts.

### 9. Distribution of Tokens Routing

The distribution of tokens routed to each expert ID within each MoE layer was examined to provide insights into load balancing. Expert usage is aggregated and normalized to produce a per-layer expert usage heatmap, which shows how balanced the routing is across the model. A Heatmap of expert token counts per layer, useful for identifying overloaded or underutilized expert, is used as a visualization tool.

These metrics allow us to deeply inspect model routing behavior. Figures 20–22 provide heatmaps of token routing frequency under different gating mechanisms, where color intensity denotes the number of tokens routed to a specific expert at each layer.

The Top-k Routing (Fig. 20) results in a highly uneven distribution of token traffic. Only a small subset of experts per layer is selected, with the rest remaining virtually unused. This is a direct consequence of rigid number of expert selection, which, while efficient (as can be seen from lowest FLOPs consumed), leads to poor expert usage. The consistently bright regions in the heatmap indicate repeated selection of the same experts across batches.

In contrast, Top-p Routing (Fig. 21) activates experts based on a cumulative probability threshold, allowing for a more stochastic and input-dependent routing. The heatmap reflects this: nearly all experts across most layers are utilized, leading

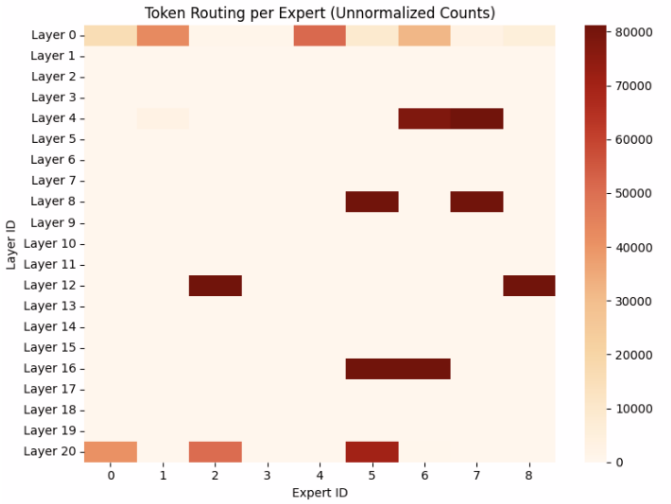



Fig. 20. Token routing counts for Top-k routing policy. Each cell represents the total number of tokens routed to a specific expert (x-axis) in a given layer (y-axis).

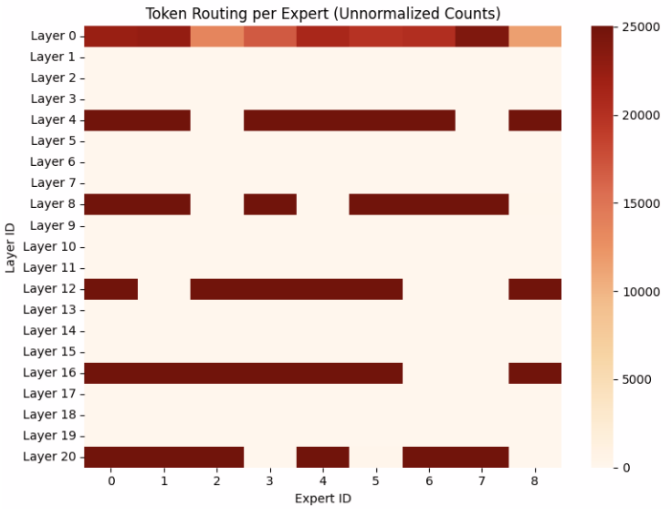



Fig. 21. Token routing counts for Top-p routing policy. Each cell represents the total number of tokens routed to a specific expert (x-axis) in a given layer (y-axis).

to a more even and diverse expert engagement pattern. This results in improved load balancing and gives the potential for the routing gate to adapt to different input distributions.

Top-any Routing (Fig. 22) serves as an intermediate strategy. The heatmap reveals moderate diversity: most experts receive some traffic, but a few are disproportionately favored.

These visualizations emphasize the profound effect routing policies have on expert utilization. While Top-k ensures low compute cost, it sacrifices diversity and load balancing. Top-p maximizes routing flexibility and model utilization, at the cost of increased computation. Top-any strikes a middle ground. Selecting a routing policy should therefore reflect the deployment context: compute-bound environments may

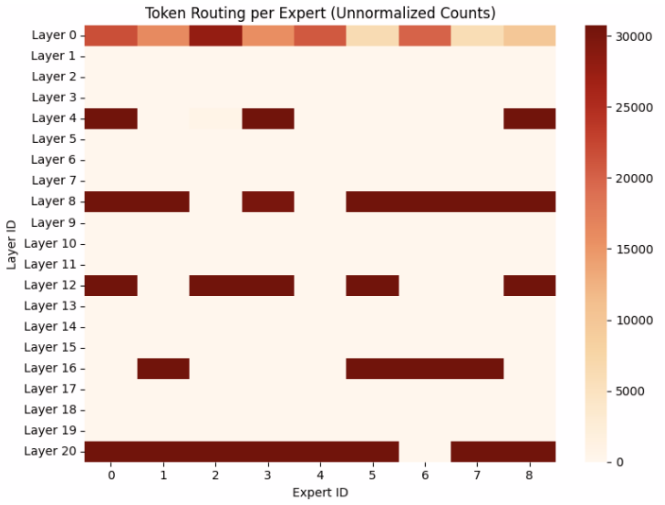



Fig. 22. Token routing counts for Top-any routing policy. Each cell represents the total number of tokens routed to a specific expert (x-axis) in a given layer (y-axis).

prefer Top-k, whereas Top-p is better suited for applications requiring maximum expressiveness and adaptivity.

## IV. Discussion

### A. Summary of Findings

This work presented a comparative evaluation of sparsely activated transformer models using various Mixture-of-Experts (MoE) routing strategies, including `Top-k`, `Top-p`, `Top-any`, and an entropy-adaptive gating mechanism. Our results demonstrate that all MoE-based models significantly outperform the dense baseline in terms of both cross-entropy loss and perplexity, achieving up to a 66% reduction in uncertainty during next-token prediction. The `Top-any` and `Top-p` strategies showed the strongest performance gains, albeit at the cost of higher memory and compute usage.

From an efficiency standpoint, `Top-k` consistently achieved the lowest latency and compute overhead, highlighting its suitability for resource-constrained deployment. The results also suggest that routing flexibility correlates with better performance, but incurs trade-offs in hardware demand and expert load balancing. However, further research is needed to confirm this result because some of the behaviours, especially with input prompt complexity correlation, were unexpected.

### B. Interpretations and Insights

Qualitative analysis of expert routing behavior revealed several important patterns. Fixed routing strategies like `Top-k` led to expert monopolization, underutilizing model capacity and showing no adaptability to input complexity. In contrast, `Top-p` and `Top-any` routing demonstrated higher expert usage.

Correlational analysis between input entropy and expert activation showed a strong positive relationship only with `Top-p` with 4 experts, suggesting that adaptive routing policies are able to dynamically adjust computational effort based on input complexity. With further research, MoE models can not only scale model capacity efficiently but also provide a mechanism to condition computation on the difficulty of the input.

### C. Lessons Learned

Profiling FLOPs, memory, and latency per token proved crucial to understanding the true operational cost of each routing strategy. Visualizing routing distributions and correlating them with token entropy was especially valuable for diagnosing model behavior and assessing dynamic routing quality.

We faced a number of engineering and resource challenges. Our MoE architecture required approximately 30GB of GPU memory for finetuning, but the AWS `g5.xlarge` instances we initially used only provided 20GB. This discrepancy led to frequent out-of-memory (OOM) errors, forcing us to manually clear GPU cache and restart training processes repeatedly. These interruptions significantly slowed down experimentation and model convergence.

Additionally, hyperparameter tuning proved especially difficult, as modifying MoE parameters—such as the number of experts or routing thresholds—typically required retraining the entire model from scratch. This constraint limited our ability to perform wide sweeps or iterative refinement.

Exposing and tuning internal MoE components also required custom logic. For instance, implementing per-expert thresholds in `Top-any` routing or adjusting entropy thresholds for adaptive gating involved manual changes to the model internals. Making these components configurable in a generalizable way took significant effort, especially given the lack of off-the-shelf support in standard libraries.

Together, these challenges highlighted the practical barriers to deploying MoE models in low-resource environments and emphasized the need for better tooling around dynamic sparse architectures.

### D. Future Work

Several promising directions remain open for future research. First, integrating reinforcement learning (RL) as a gating mechanism offers a principled way to treat expert selection as a learned policy. Since expert routing inherently defines a policy over a discrete action space (i.e., selecting one or more experts), RL methods such as REINFORCE can be employed to learn this policy based on a reward function. The reward can jointly optimize for predictive accuracy and expert load balancing—two often competing objectives. To reduce the high variance typically associated with REINFORCE, variance-reduction techniques (e.g., using a baseline) could be incorporated. A particularly effective strategy could involve first training a supervised reference model purely for accuracy. This model could then serve as a fixed reference policy, enabling KL-regularized RL to steer exploration toward better load-balanced solutions while preserving accuracy. Such a gating mechanism could be trained offline to avoid inference-time overhead, and then distilled into a lightweight router for deployment. This would offer dynamic, context-sensitive

| Model | Loss | Perplexity | Time/Token (ms) | Memory (MB) | FLOPs (G) | Experts/Token |
|---|---|---|---|---|---|---|
| Baseline (Dense) | 9.41 | 12,153 | 0.1246 | 2215.7 | 2234.6 | N/A |
| Top-k (E=8) | 8.35 | 4234.9 | 0.0718 | 4867.3 | 787.1 | 2.00 |
| Top-p (E=8) | 8.32 | 4124.3 | 0.1054 | 5385.3 | 1227.0 | 6.49 |
| Top-any (E=8) | 8.31 | 4079.4 | 0.1083 | 5321.9 | 1108.9 | 5.28 |
| Entropy-Adaptive | 8.32 | 4111.8 | 0.1263 | 5318.4 | 983.2 | N/A |
| Top-k (E=4) | – | – | – | – | – | 2.00 |
| Top-p (E=4) | – | – | – | – | – | 4.86 |

TABLE IX
PERFORMANCE SUMMARY ACROSS ROUTING CONFIGURATIONS. ALL MOE MODELS OUTPERFORM THE DENSE BASELINE IN PERPLEXITY AND LOSS, WITH VARYING TRADE-OFFS IN COMPUTE AND MEMORY.

expert selection with minimal runtime cost—advancing the flexibility and efficiency of MoE systems.

Second, extending these evaluations to multilingual, vision-language, or long-context benchmarks could better validate our results. Finally, hardware-aware routing strategies that adapt based on memory availability or parallelism constraints represent a practical and impactful next step.

## V. RELATED WORKS

*Mixture of Experts Meet Instruction Tuning [6]*

This paper implements MoE at scale by proposing sparsely gated MoE layers, where only a few expert sub-networks are activated per forward pass using a learned gating mechanism. It forms the foundation for our project, particularly the use of sparse expert selection strategies. We build on this concept by investigating alternative routing strategies—Top-K, Top-P, and Top-Any—to optimize expert selection at inference time.

*Switch Transformers [9]*

This work scales Transformers to trillion-parameter levels by replacing dense FFNs with sparsely gated "Switch" MoE layers, activating only one expert per token. This significantly reduces training FLOPs and memory usage while maintaining performance. We adopt their single-expert-per-token setup as a baseline and expand the routing mechanism by testing Top-K and Top-P to improve load balancing and efficiency.

*DeepSeek MoE [8]*

DeepSeek MoE introduces finer sub-experts and a shared expert pool, enabling the router to select more specialized combinations per token. Achieving LLaMA-2 performance with 40% compute, this work inspires our Top-P and Top-Any routing strategies aimed at improved load balancing through more flexible expert activation.

*LLaMA-MoE [10]*

This paper demonstrates how to retrofit a dense LLaMA-2 7B model into a sparse MoE using lightweight expert partitioning and Top-K gating, followed by continual pre-training. Their efficient adaptation method motivates our exploration of alternative routing mechanisms like Top-Any to further optimize sparse conversions.

*Harder Tasks Need More Experts [3]*

This paper shows that confidence-based gates outperform fixed Top-2 routing in both accuracy and efficiency, supporting the idea that different tokens need different expert capacities. Our Top-Any and entropy-threshold-based routers build on this principle, aiming for dynamic capacity allocation to enhance inference-time performance.

*Dynamic Mixture of Experts (DynMoE) [2]*

DynMoE combines Top-Any routing with a training-time scheduler that can grow or prune experts, enabling tokens to activate only the capacity they need. It achieves Switch-level performance while activating fewer than 6% of parameters. We extend their Top-Any method with entropy thresholds and fine-tuning to explore smoother load balancing and reduced compute.

*Mixture of Experts with Expert Choice Routing [11]*

Instead of tokens choosing their top-k experts, this paper proposes allowing experts to select their top-k tokens. This "Expert Choice" gating reduces load imbalance and accelerates convergence. While our work focuses on token-based routing, this paper offers a complementary perspective that may inform future hybrid routing schemes.

*MoEfication [7]*

MoEfication repurposes pretrained dense Transformers by splitting FFN layers into functional partitions and adding lightweight routers, transforming them into sparse models without additional parameters. This retrofit approach aligns with our goals and motivates our exploration of routing strategies like Top-P to maximize the performance of MoEfied architectures.

## VI. CONCLUSION

In this paper, we explored the design, implementation, and comparative evaluation of multiple expert routing strategies in Mixture of Experts (MoE) models—namely, Top-k, Top-p, Top-any, and Entropy-Adaptive routing—within a GPT-2 architecture. Our primary goal was to assess how different gating mechanisms influence model performance, compute efficiency, and expert utilization.

Our results demonstrate that all MoE configurations substantially outperform the dense baseline in terms of both average loss and perplexity. Notably, Top-any and Top-p routing strategies yielded the best overall language modeling performance, achieving the lowest loss and perplexity scores. However, these gains came at the cost of increased memory usage and inference-time compute, highlighting the intrinsic trade-off between model expressiveness and computational efficiency.

Through a comprehensive set of analyses—including expert usage heatmaps, input-entropy correlations, and per-layer routing visualizations—we observed that dynamic routing mechanisms like Top-p and Top-any enable more balanced and adaptive expert utilization. These strategies allow the model to scale its computational effort based on the complexity of the input, an ability that fixed Top-k routing lacks.

Our study reaffirms that routing flexibility plays a crucial role in unlocking the full potential of sparse expert architectures. Our findings serve as a design guide for researchers and practitioners seeking to deploy efficient yet powerful sparse language models.

## REFERENCES


[1] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," *OpenAI Blog*, 2019.

[2] W. Jiang, Y. Zhang, Y. Yuan, W. Wang, Z. Huang, W. Zhao, S. Zhang, and X. Song, "Dynamic Mixture of Experts: An Auto-Tuning Approach for Efficient Transformer Models," *arXiv preprint*, arXiv:2405.14297, May 2024.

[3] Q. Huang, C. Qin, Y. J. Kim, and D. Alistarh, "Harder Tasks Need More Experts: Dynamic Routing in MoE Models," *arXiv preprint*, arXiv:2403.07652, March 2024.

[4] W. Zhang, Z. Zhang, H. Wang, Y. Wu, and C. Xu, "SEER-MoE: Sparse Expert Efficiency through Regularization for Mixture-of-Experts," *arXiv preprint*, arXiv:2404.05089, Apr. 2024.

[5] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," arXiv preprint, arXiv:1701.06538, January 2017.

[6] Q. Zheng, H. Wang, Z. Zhang, Y. Song, Z. Liu, and H. Ma, "Mixture of Experts Meet Instruction Tuning," *arXiv preprint*, arXiv:2305.14705, May 2023.

[7] Y. Zhang, S. Sun, X. Qiu, L. Hou, Y. Liu, J. Liu, J. Liu, M. Zhou, and W. Chen, "MoEfication: Scalable and Efficient MoE Training for Pre-trained Transformers," *arXiv preprint*, arXiv:2110.01786, October 2021.

[8] DeepSeek AI, "DeepSeekMoE: Scaling Language Models with Mixture-of-Experts in All Layers," *arXiv preprint*, arXiv:2401.06066, January 2024.

[9] W. Fedus, B. Zoph, and N. Shazeer, "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity," *arXiv preprint*, arXiv:2101.03961, January 2021.

[10] W. Zhao, W. Tang, Y. Deng, S. Xiao, S. Wang, X. Liang, X. Lin, S. Gu, and X. Song, "LLaMA-MoE: Efficiently Retrofitting Sparse Mixture of Experts into Pretrained LLaMA Models," *arXiv preprint*, arXiv:2406.16554, June 2024.

[11] M. Lewis, D. Yarats, S. Shleifer, Y. Lu, S. Narang, J. Xu, A. Baevski, J. Ainslie, G. Wenzek, X. Chen, and others, "Mixture of Experts with Expert Choice Routing," *arXiv preprint*, arXiv:2202.09368, February 2022.