Faster MoE LLM Inference for Extremely Large Models

Haoqi Yang[1*] **Luohe Shi**[1*] **Qiwei Li**[1] **Zuchao Li**[2†] **Ping Wang**[3] **Hao Huang**[1†] **Hai Zhao**[4]
[1]School of Computer Science, Wuhan University, Wuhan, China
[2]School of Artificial Intelligence, Wuhan University, Wuhan, China
[3]School of Information Management, Wuhan University, Wuhan, China
[4]School of Computer Science, Shanghai Jiao Tong University, Shanghai, China
{yanghq, shiluohe, qw-line, zcli-charlie, wangping, haohuang}@whu.edu.cn
zhaohai@cs.sjtu.edu.cn

Abstract

In fine-grained sparse Mixture-of-Experts (MoE) models, a large pool of specialized experts replaces a small homogeneous set, shifting performance and throughput to be governed by inference-time expert activation. Yet most existing optimization recipes implicitly assume a fixed activation budget (e.g., a constant Top-k per layer), whose behavior in fine-grained MoEs is poorly understood. We first characterize runtime skipping strategies, quantifying the accuracy–efficiency trade-off of (i) uniform fixed activation and (ii) static layer-wise Top-k allocation found by search. Our analysis reveals that static skipping can already provide substantial throughput gains, but optimal static schedules vary significantly across models and routing mechanisms. We therefore introduce Adaptive Skipping with Entropy-Penalized Thresholding (ASET), a training-free policy that adapts token-level activation using router confidence and entropy while remaining within the model's original budget. Across the fine-grained MoEs we study, static skipping policies yield 10–78% throughput gains with minimal performance degradation, including $\geq$10% improvement on DeepSeek-V3 without measurable loss. On the OLMoE testbed, ASET yields a Pareto frontier between average activation and task quality. Overall, these results identify expert skipping as a practical lever for faster fine-grained MoE inference, with adaptive activation helping when fixed budgets are too rigid.

1 Introduction

Mixture-of-Experts (MoE) models sparsify the feed-forward network (FFN), which contains most parameters in modern large language models, by partitioning it into multiple experts and activating only a small subset per token (Shazeer et al., 2017; Fedus et al., 2022). This design enables scaling model capacity without proportional growth in training and inference compute. However, as MoEs become a mainstream backbone for open-source LLMs (e.g., Mixtral and DeepSeek), their inference efficiency becomes a first-order deployment bottleneck.

Most existing inference optimizations were developed for coarse-grained MoEs (e.g., selecting 2 experts out of 8), where reducing the number of activated experts is inherently limited and global expert pruning/merging is often viable due to higher inter-expert similarity. In contrast, recent fine-grained MoEs (Dai et al., 2024) employ larger expert pools and activate more experts per token, expanding the optimization space while weakening the assumptions behind coarse-grained techniques. However, a practical inference question remains. These models typically enforce a rigid activation budget (e.g., fixed Top-k) for every token, even though token difficulty can vary substantially: some tokens require the consensus of multiple experts, whereas others can be resolved by only a few. For fine-grained MoE serving, should expert activation follow a universally fixed budget, or be adjusted to token-level uncertainty?

A natural starting point is *runtime expert skipping*: reducing the number of activated experts during inference. We systematically evaluate skipping strategies in fine-grained MoEs, including uniform fixed activation and searched layer-wise static allocation. Our analysis shows that static skipping can provide substantial throughput gains, but that the best fixed-budget rule is brittle: even within the family of static policies, the preferred layer-wise allocation pattern is model-dependent. For instance, the allocation strategy that works best for DeepSeek-V2-Lite fails to transfer to DeepSeek-V3, and vice versa. These reversals indicate that expert skipping is a useful efficiency lever, but no single static Top-k pattern appears

[*]Equal contribution.
[†]Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 43133–43151

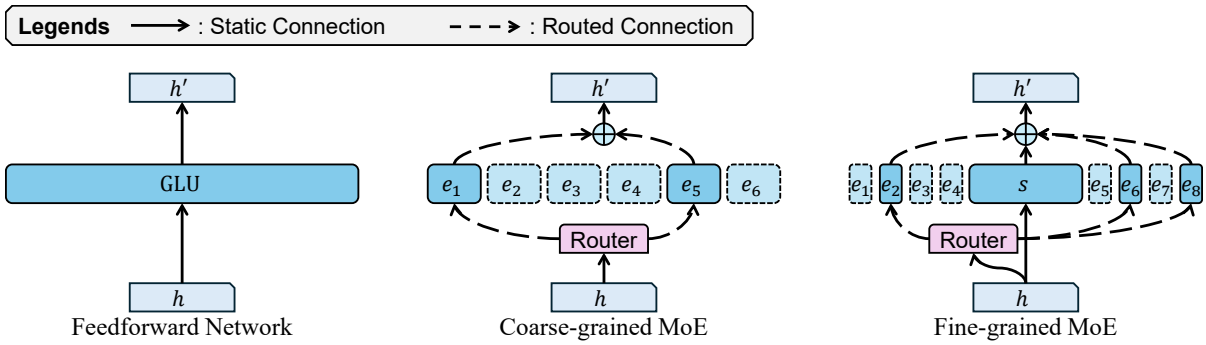

Figure 1: A comparison of dense FFN, coarse-grained MoE, and fine-grained MoE. Fine-grained MoE increases the expert pool size and the number of activated experts, which expands the inference optimization space but reduces inter-expert substitutability.

universal across fine-grained MoEs.

These findings suggest that expert activation is better understood as a *token-level compute allocation* problem. Fixed Top-k treats all tokens identically, even though routers under different architectures (e.g., Softmax or Sigmoid) can produce sharply different routing profiles, ranging from concentrated, high-confidence distributions to diffuse, ambiguous ones. A single activation budget therefore need not be equally appropriate for every token. This motivates inference rules that adapt the activation count to token-level uncertainty rather than relying on a universally fixed budget.

Based on this view, we propose **Adaptive Skipping with Entropy-Penalized Thresholding (ASET)**[1], a training-free inference policy for adaptive expert activation in fine-grained MoE serving. For each token, ASET derives a token-specific cumulative-confidence threshold from router confidence and the entropy of a compact candidate distribution, then retains the smallest expert subset whose cumulative confidence reaches a threshold defined relative to the router's original Top-n_a confidence mass. Under this formulation, fixed Top-k and fixed-threshold rules emerge as simpler non-adaptive special cases in which the activation rule does not respond to token-level uncertainty. ASET therefore remains within the same training-free, within-budget expert-skipping regime as our static analyses, while replacing a fixed activation rule with token-adaptive control over the retained expert subset. This setup lets us compare token-adaptive activation with static skipping under the same deployment constraint, asking whether token-level control can use the original activation budget more effectively.

In summary, this paper makes three contributions:

- We systematically characterize inference-time expert skipping in fine-grained MoEs, quantifying the accuracy–efficiency trade-off of uniform fixed activation.
- We develop and evaluate a layer-wise Top-k scheduling family, which attains strong efficiency–quality trade-offs within fixed-budget policies yet exhibits model-specific optimal patterns that do not reliably transfer.
- We propose ASET, an uncertainty-adaptive, token-wise activation mechanism, and characterize its Pareto trade-off between average activation and task quality on OLMoE, a controllable fine-grained MoE testbed.

2 Background and Related Work

2.1 FFN and MoE

A feed-forward network (FFN) typically takes the form of an MLP or a GLU, as shown in Equation 1, where $\mathrm{ACT}(\cdot)$ denotes the activation function and $\otimes$ denotes the element-wise product. Throughout this paper, we use the GLU form to represent FFNs because it is more common in modern LLMs (Shazeer, 2020; Chowdhery et al., 2023).

$$\begin{aligned}\mathrm{MLP}(h) &= W_{\mathrm{d}} \cdot \mathrm{ACT}\left(W_{\mathrm{u}} \cdot h\right) \\ \mathrm{GLU}(h) &= W_{\mathrm{d}} \cdot \left(\mathrm{ACT}\left(W_{\mathrm{u}} \cdot h\right) \otimes \left(W_{\mathrm{g}} \cdot h\right)\right)\end{aligned} \tag{1}$$

A mixture-of-experts (MoE) layer can be written as a weighted combination of multiple GLUs, where the routing weights r are produced by a linear router. We denote the corresponding router logits by r'. We use $F_r(\cdot)$ to denote any transforma-

[1]Code is available at https://github.com/brinenick511/ASET.git.

tion applied before Top-k selection, for example for load balancing, and $F_w(\cdot)$ to denote the normalization that produces the final routing weights. An MoE layer with n_e experts and n_a activated experts is given in Equation 2.

$$\begin{aligned} \mathrm{MoE}(h) &= \sum_{i=1}^{n_e} r_i \cdot \mathrm{GLU}_i(h) \\ r' &= W_{\mathrm{r}} \cdot h, \\ r_i &= \begin{cases} F_w(r'_i), & i \in \mathrm{TopK}\big(F_r(r'), n_a\big) \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{2}$$

2.2 LLM Serving Efficiency

LLM serving efficiency is typically characterized by deployment-level objectives, including throughput, latency, and SLO-oriented metrics Wang et al., 2024. Among these objectives, a major line of work targets throughput by improving batching, memory management, and hardware utilization. Representative systems such as Orca (Yu et al., 2022) and vLLM (Kwon et al., 2023), often discussed through Roofline-style analysis (Williams et al., 2009), highlight effective batch size as a central lever for high-throughput decoding. Techniques such as continuous batching and chunk attention (Ye et al., 2024) follow this line by improving the efficiency of processing shared-prefix KV-cache segments.

Recent work also explores orthogonal acceleration directions, including speculative decoding under large-batch regimes (Shi et al., 2026). At the MoE level, prior work has studied token-adaptive routing under different deployment assumptions, including training-integrated approaches with null experts (Zeng et al., 2024). In contrast, we study training-free inference-time expert skipping in fine-grained MoEs, where activation is reduced from the original routing outcome rather than by modifying the routing mechanism itself. For this reason, throughput is the primary metric in our controlled setting: under a fixed decoding setup, it directly reflects how changes in expert activation affect serving capacity.

2.3 Model Pruning

A related but distinct route to efficiency is model pruning, which can improve inference efficiency with limited quality degradation and can operate at different levels of granularity. More broadly, recent work places pruning within a wider compression landscape that also includes quantization and related techniques (Ma et al., 2026). Representative methods range from layer skipping (Gromov et al., 2025) to within-layer sparsification (Frantar and Alistarh, 2023; He et al., 2024). Related serving-side compression work also targets KV-cache reduction through quantization or latent-space compression, which can alleviate memory and bandwidth pressure during decoding (Yang et al., 2025; Luohe et al., 2025).

For MoE models, related work has primarily focused on expert pruning and merging (Chen et al., 2022; Yang et al., 2024b; Xie et al., 2024; Lu et al., 2024; Lee et al., 2024; Muzio et al., 2024; Li et al., 2024; Chen et al., 2025). Such approaches are comparatively natural in coarse-grained MoEs, where experts are more homogeneous, but become more challenging in fine-grained MoEs because weaker expert substitutability makes permanent expert removal less reliable. At the same time, compared with coarse-grained MoEs, the larger expert pools and activated-expert budgets of fine-grained MoEs leave more room for a complementary runtime strategy: rather than permanently removing experts, one can keep the full expert pool intact and adapt only the activated subset at inference time. Appendix A provides a secondary empirical analysis of expert pruning.

3 Preliminaries

3.1 Notations

We previously defined d as the hidden size, d_i as the intermediate size of FFNs, n_e as the number of experts, n_a as the number of active experts per token, F_r as the modifier applied to router logits for expert selection, and F_w as the weight normalization function. We denote by L the sequence length, by $h \in \mathbb{R}^d$ the hidden state, by $H \in \mathbb{R}^{d \times L}$ the hidden-state matrix, and by $W_u, W_g \in \mathbb{R}^{d_i \times d}$ and $W_d \in \mathbb{R}^{d \times d_i}$ the GLU parameters. The memory I/O, FLOPs, and arithmetic intensity (AI) for the triplet (d, d_i, L) are given in Equation 3.

$$\begin{aligned} \mathrm{I/O}(d, d_i, L) &= 3d_i d + 2L(d + d_i) \\ \mathrm{FLOPS}(d, d_i, L) &= 6L(d_i d) \\ \mathrm{AI}(d, d_i, L) &= \frac{6L(d_i d)}{3d_i d + 2L(d_i + d)} \end{aligned} \tag{3}$$

To better evaluate the size of each state, we further define d_e as the intermediate size of experts, d_s as the intermediate size of the shared expert, and d_a as the activated intermediate size, where $d_a = d_e \times n_a$.

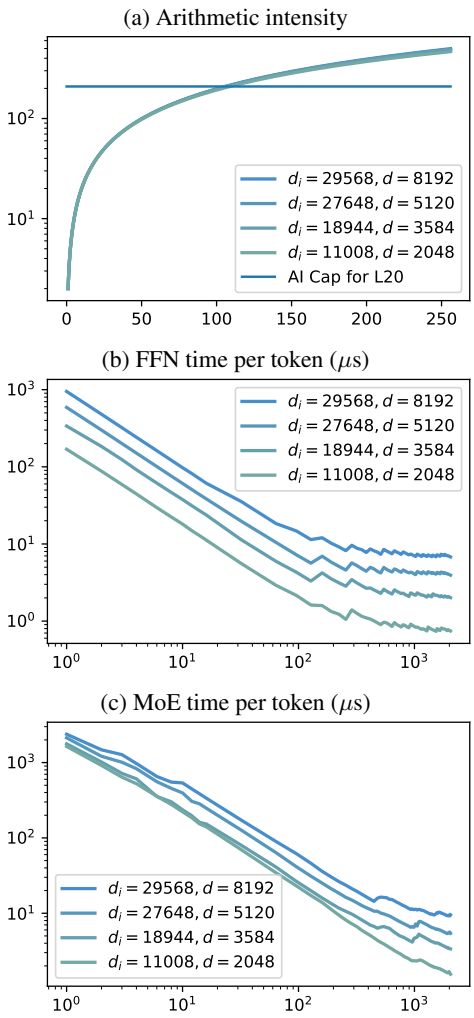

Figure 2: Simulation experiment results. X-axis represents sequence length L.

3.2 Batching Effect

The feed-forward layer constitutes the majority of model parameters, accounting for approximately 66% in earlier models and up to 88% in some modern models (Yang et al., 2024a). In a single-batch setting, it dominates computation time. During the prefill phase, all tokens in a sequence pass through the FFN simultaneously, meaning that once parameters are loaded into memory, they are reused multiple times. This amortizes the parameter loading cost across multiple tokens.

Figure 2a depicts the relationship between sequence length L and arithmetic intensity (AI), i.e., the ratio between floating-point operations and memory traffic. When the batch size is small, increasing the number of parallel tokens significantly improves system performance, as illustrated in Figure 2b, which shows the relationship between L and per-token latency (μs). Efficiency peaks at around $L = 150$. Importantly, FLOPs-based "compute fraction" is only a proxy for potential speedup. When the kernel is memory- or scheduling-bound, reducing the amount of activated parameters can yield speedups that exceed the FLOPs reduction, because it also reduces memory traffic and routing overhead.

In MoE models, tokens rarely reuse the same expert within a batch; this creates additional memory access overhead. Consequently, even without considering scheduling overhead (which can be significant), MoE is inherently not faster than FFN when operating under the same activated parameter count. We evaluated the efficiency of the MoE module across different sequence lengths, as shown in Figure 2c. Our findings indicate that the MoE module incurs higher latency and reaches its peak efficiency more slowly compared to Figure 2b. This batching disadvantage motivates inference-time expert skipping, which reduces both computation and memory traffic by activating fewer experts per token.

4 Inference-Time Expert Skipping

Compared to coarse-grained MoE, which typically activates only 2 experts, fine-grained MoE selects 6 to 8 experts from a pool of 64 to 256. This makes the activated-expert count n_a an explicit inference-time compute budget that can be re-allocated at the token level. In this section, we use controlled expert skipping as a diagnostic to study how quality and throughput respond to different activation budgets.

4.1 Static Uniform Top-k Reduction

The fine-grained paradigm therefore makes **expert skipping** a viable way to reduce and re-allocate this budget, whereas this approach is largely impractical in coarse-grained models where the baseline of 2 activated experts offers minimal room for reduction. We begin with a uniform policy in which every MoE layer uses the same reduced activation count, because it provides the simplest controlled baseline for measuring how quality and throughput respond to a lower global activation budget. This setting separates the question of *how much* expert compute to retain from the question of *where* to retain it across layers. Unlike expert pruning, which aims to reduce the memory footprint, expert skipping keeps the full expert pool intact and changes only the activated subset at run-

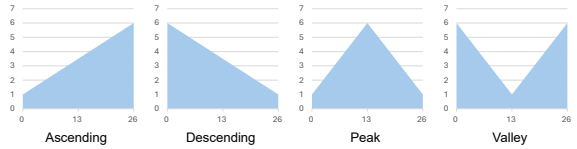

Figure 3: Structure shapes in Section 4.

time, making it directly relevant to serving-time compute allocation.

4.2 Static Layer-wise Top-k Scheduling

Uniform skipping assumes that all MoE layers use the same activated capacity. We generalize this setting to a static layer-wise policy that allows different activated-expert counts across layers under a constrained overall budget, thereby asking whether quality depends not only on how much expert compute is retained, but also on where that budget is placed. Rather than proposing a universal layout, we use structure search as a diagnostic to measure how sensitive model quality is to the layer-wise placement of activation reductions. The parameterization is intentionally low-dimensional: instead of enumerating all possible schedules, we construct a small family of interpretable shapes that shift the budget toward earlier, middle, or later layers. For each configuration, we report performance against the average activated experts $\bar{n}_a$ (averaged across layers), enabling fair comparisons across layouts with matched overall budgets. Specifically, we define expert allocation using a four-tuple (b, h, e, p) that parameterizes this shape family, where:

- The first layer selects b experts, i.e., $n_a(1) = b$.
- The p-th layer selects h experts, i.e., $n_a(p) = h$.
- The last layer selects e experts, i.e., $n_a(-1) = e$.
- For layers not specified, expert counts are determined through linear interpolation.

This formulation allows us to explore various expert allocation patterns, including ascending, descending, peak, and valley-shaped distributions, depicted in Figure 3.

4.3 Adaptive Skipping with Entropy-Penalized Thresholding

Conventional MoE gating activates a fixed number (n_a) of experts per token, regardless of whether the router distribution for that token is sharply concentrated or broadly diffuse. In fine-grained MoEs, such a fixed-budget rule can allocate more compute than necessary to high-confidence tokens while becoming restrictive for more ambiguous ones. This observation motivates a token-level view of expert activation in which the activation budget can vary with token difficulty.

A unifying view. We cast expert skipping as token-level compute allocation: given sorted routing scores p', we choose the smallest k such that the retained probability mass exceeds a token-dependent threshold $\tau(x)$, $k(x) = \arg\min_k \left(\sum_{i=1}^{k} p_{(i)} \geq \tau(x)\right)$. This view separates *what signal* defines token difficulty (e.g., entropy) from *how* that signal is converted into an activation budget. Fixed Top-k can be viewed as a constant-budget rule, while fixed-threshold or Top-p-style criteria correspond to a token-invariant threshold. ASET differs by adapting this threshold to token-level ambiguity while anchoring the final decision to the router's original Top-n_a confidence mass, thereby remaining strictly within the original activation budget.

Our adaptive skipping design is guided by three considerations. First, cumulative confidence provides a more flexible control signal than a fixed activation count. Second, token ambiguity is reflected not only in retained probability mass but also in the shape of the routing distribution, which motivates using entropy as an additional signal. Third, training-time regularization can make the lower-probability tail of the router distribution less informative for inference-time control, suggesting that ambiguity should be measured on a compact candidate subset.

Guided by these considerations, we introduce **Adaptive Skipping with Entropy-Penalized Thresholding (ASET)**, which dynamically calibrates expert activation while remaining within the original activation budget. Let the gating output be a probability distribution $p \in \mathbf{R}^{n_e}$, where $p = \text{softmax}(r)$ for the router logits r. We denote the probabilities sorted in descending order as $p' = (p_{(1)}, p_{(2)}, \ldots, p_{(n_e)})$. ASET then unfolds in three stages:

1. Probabilistic Relevance Masking Recognizing that training-time regularizers (e.g., load-balancing) can make the low-probability tail of the router distribution less informative for inference-time control, we compute uncertainty and thresholds on a compact candidate set rather than on the full distribution. Specifically, we define the candidate set $\mathcal{C}$ as the smallest prefix of sorted experts whose cumulative probability mass reaches $\theta_{\max}$.

$$n_w = \arg\min_k \left(\sum_{i=1}^{k} p_{(i)} \geq \theta_{\text{max}} \right) \quad (4)$$

The resulting candidate set is $\mathcal{C} = \{e_{(1)}, \ldots, e_{(n_w)}\}$, with the corresponding probability sub-vector $p_c = (p_{(1)}, \ldots, p_{(n_w)})$. By restricting ambiguity estimation to this focused subset, ASET reduces the influence of tail mass and derives a cleaner signal for adaptive thresholding. For simplicity, we set $\theta_{\text{cand}} = \theta_{\text{max}}$ in all experiments unless stated otherwise.

2. Measuring Gating Decision Ambiguity The second stage quantifies the router's decision ambiguity within the candidate set $\mathcal{C}$ using Shannon Entropy. To compute a valid entropy, we first renormalize the probabilities in p_c to form a proper probability distribution $\hat{p}_c$:

$$\hat{p}_{c,i} = \frac{p_{(i)}}{\sum_{j=1}^{n_w} p_{(j)}} \quad \forall i \in \{1, \ldots, n_w\} \quad (5)$$

We then compute the Shannon Entropy of this distribution, $H(\hat{p}_c)$:

$$H(\hat{p}_c) = -\sum_{i=1}^{n_w} \hat{p}_{c,i} \log_2 \hat{p}_{c,i} \quad (6)$$

When $n_w = 1$, the router decision is unambiguous within $\mathcal{C}$; we define $H_{\text{norm}}(\hat{p}_c) = 0$ by convention. Finally, this entropy is normalized to produce a scale-invariant ambiguity metric, H_{norm}. This is achieved by dividing by the maximum possible entropy for n_w candidates, $\log_2(n_w)$:

$$H_{\text{norm}}(\hat{p}_c) = \begin{cases} 0, & n_w = 1, \\ \frac{H(\hat{p}_c)}{\log_2(n_w)}, & n_w > 1. \end{cases} \quad (7)$$

This metric, bounded in $[0, 1]$, quantifies the signal's structure: a value approaching 0 corresponds to a low-entropy, peaky signal, while a value approaching 1 indicates a high-entropy, diffuse signal.

3. Adaptive Threshold Computation In the final stage, an adaptive threshold θ_{dyn} is calculated, non-linearly modulated within the range $[\theta_{\text{min}}, \theta_{\text{max}}]$, guided by the ambiguity H_{norm} and the interpolation curvature parameter $\gamma > 1$.

$$\theta_{\text{dyn}} = \theta_{\text{min}} + (\theta_{\text{max}} - \theta_{\text{min}}) \cdot (H_{\text{norm}})^{\gamma} \quad (8)$$

The curvature induced by the interpolation curvature parameter penalizes ambiguity, ensuring θ_{dyn} remains close to the aggressive lower bound θ_{min} unless the router's belief state is exceptionally diffuse. A key insight of ASET is that this threshold should be relative to the router's baseline confidence in its original Top-n_a choices. We define this baseline confidence scale as $S = \sum_{i=1}^{n_a} p_{(i)}$. This anchors the adaptive threshold to the router's training-time activation budget: since $\theta_{\text{dyn}} \in [\theta_{\text{min}}, \theta_{\text{max}}] \subseteq [0, 1]$, we always have $S \cdot \theta_{\text{dyn}} \leq S$, implying $k_{\text{final}} \leq n_a$. In other words, ASET only *skips* experts relative to the original Top-n_a plan, never expanding the budget. The final number of experts to activate, k_{final}, is determined using this adaptive threshold on the original sorted probabilities p':

$$k_{\text{final}} = \arg\min_k \left(\sum_{i=1}^{k} p_{(i)} \geq S \cdot \theta_{\text{dyn}} \right) \quad (9)$$

This design makes ASET sensitive to router confidence while preserving a principled allocation of expert compute within the original activation budget.

5 Experiments

5.1 Evaluation Setup

We evaluate three representative fine-grained MoE models: DeepSeek-V2-Lite-Chat, DeepSeek-V3, and OLMoE-1B-7B-0125-Instruct (Muennighoff et al., 2024). Unless otherwise specified, efficiency is reported as decoding throughput (generated tokens per second), excluding the prefill stage; end-to-end throughput is explicitly identified when reported.

We evaluate model quality on several benchmark datasets: ARC (Easy and Challenge, Clark et al., 2018), BoolQ (Clark et al., 2019), OpenBookQA (OBQA, Mihaylov et al., 2018), RTE (Bentivogli et al., 2009), and Winogrande (Sakaguchi et al., 2021). Unless otherwise specified, the reported performance score is the average over these benchmarks, where our aggregation yields a random-guess baseline of approximately 36.

5.2 Implementation Details

Measured throughput is highly sensitive to the input/output length configuration (Appendix B). To control for this factor, unless otherwise specified, all efficiency experiments use 1024 randomly sampled input tokens and 1024 generated output to-

Algorithm 1: ASET (per token)

Input : Router probabilities $p \in \mathbb{R}^{n_e}$, training budget n_a, thresholds $\theta_{\text{cand}}, \theta_{\min}, \theta_{\max}$, coefficient γ

Output : Activation count k_{final}

Sort p in descending order to obtain $p_{(1)} \geq \cdots \geq p_{(n_e)}$;

$n_w \leftarrow \min\left\{k \mid \sum_{i=1}^{k} p_{(i)} \geq \theta_{\text{cand}}\right\}$;

$\hat{p}_{c,i} \leftarrow p_{(i)} \big/ \sum_{j=1}^{n_w} p_{(j)}$;

$H \leftarrow -\sum_{i=1}^{n_w} \hat{p}_{c,i} \log_2 \hat{p}_{c,i}$;

$H_{\text{norm}} \leftarrow \begin{cases} 0, & \text{if } n_w = 1 \\ H/\log_2(n_w), & \text{otherwise} \end{cases}$;

$\theta_{\text{dyn}} \leftarrow \theta_{\min} + (\theta_{\max} - \theta_{\min}) \cdot (H_{\text{norm}})^{\gamma}$;

$S \leftarrow \sum_{i=1}^{n_a} p_{(i)}$;

$k_{\text{final}} \leftarrow \min\left\{k \mid \sum_{i=1}^{k} p_{(i)} \geq S \cdot \theta_{\text{dyn}}\right\}$;

return k_{final}

kens. This 1024/1024 setting helps control variation in prefill fraction, which can otherwise affect end-to-end throughput through differences in parallelism, and approximates a steady-state long-form decoding workload under moderate-to-high concurrency. The construction is intended for controlled efficiency comparison rather than reproducing the length distribution of any specific benchmark. Table 3 illustrates how the input/output ratio alone can substantially change the measured throughput.

We profile throughput for DeepSeek-V2-Lite and DeepSeek-V3 with `sglang` using `sglang.bench`. DeepSeek-V2-Lite experiments run on NVIDIA A800 80GB GPUs, while DeepSeek-V3 experiments run on NVIDIA H200 141GB GPUs. The microbenchmarks in Figure 2 are implemented in PyTorch with `torch.compile`; additional platform details are provided in Appendix C–D.

5.3 Efficiency

We measure serving throughput under uniform expert skipping, where all layers use the same activated-expert count $n_a \in \{2, \ldots, n_a^{\text{orig}}\}$ while retaining the full expert pool. Based on this setup, we further evaluated the models under different request loads to analyze their performance.

Here, concurrency denotes the number of simultaneous in-flight requests, and throughput is measured as generated tokens per second under the same decoding setup (see Appendix E). Our experiment results are depicted in Figure 4.

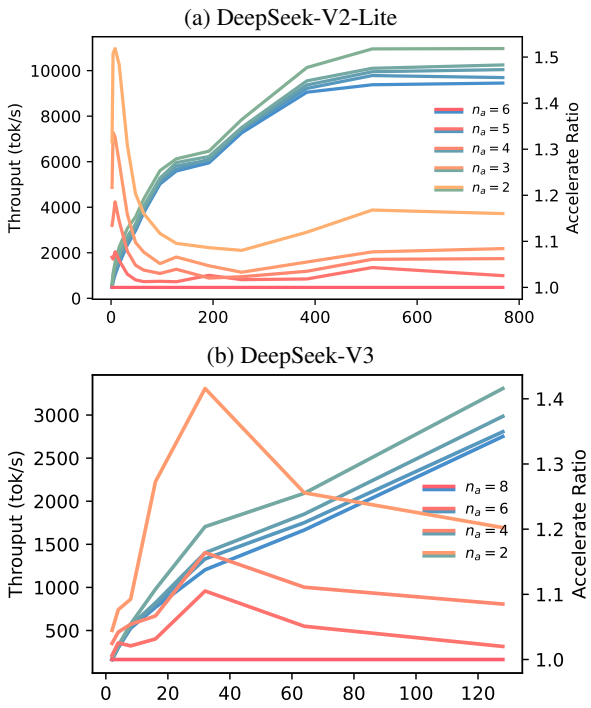

Figure 4: Throughput (generated tokens/s) under different activated-expert counts n_a with all experts retained. X-axis: concurrency (simultaneous in-flight requests); speedup is reported relative to n_a^{orig}.

Across concurrency levels, reducing the activated-expert count n_a has little effect on the concurrency required to reach peak throughput, but it substantially changes the shape of the speedup curve. This pattern indicates that the benefit of expert skipping is serving-regime dependent rather than determined by FLOPs reduction alone.

Notably, we observe substantial acceleration at both low and high concurrency levels, while the acceleration effect is limited at moderate concurrency. In fact, at low concurrency, the speedup ratio can reach 50% (when $n_a = 2$), exceeding the FLOPs-based compute-fraction bound ($d_a/(d_s + d_a) \approx 43.6\%$; excluding shared components). This is because at low concurrency, the system is memory I/O-bound, and reducing n_a immediately lowers the required parameter loading, leading to a higher proportion of acceleration. At moderate concurrency, the system remains memory I/O-bound, but since a sufficient number of tokens are processed simultaneously, reducing n_a does not materially change the concurrency range at which peak throughput is reached. At high concurrency, the system shifts to a compute-bound

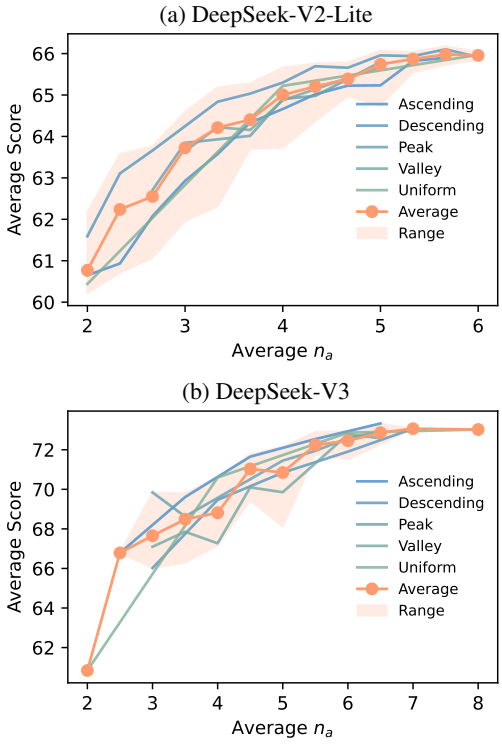

Figure 5: Performance versus average activated experts $\bar{n}_a$ under different layer-wise activation layouts (e.g., ascending/descending/peak/valley)

regime, where reducing n_a lowers computational demands, thereby increasing throughput. In this case, the throughput gain aligns more closely with the compute reduction ratio. In expert-parallel deployments where experts from the same MoE layer are partitioned across devices, the same reduction in activated experts would also reduce cross-device communication for expert dispatch and combine.

5.4 Performance under Static Layer-wise Schedules

We aim to identify how model quality depends on the layer-wise placement of a fixed activation budget. Concretely, we compare static schedules that match the total number of activated experts per token but redistribute that budget across layers, allowing us to isolate whether some parts of the network are more sensitive to expert reduction than others.

Figure 5 presents our results. For softmax-based models such as DeepSeek-V2-Lite, as shown in Figure 5a, we observe that even relatively aggressive expert skipping does not lead to a significant degradation in performance. On average, reducing the number of active experts from the full set to only two results in a performance drop of approximately 7.5%, while in the best-case scenario, the performance loss is limited to around 6%. Moreover, when an average of 3.3 experts is retained, the performance degradation remains within 1%.

In DeepSeek-V3, as shown in Figure 5b, we observe a similar performance curve when reducing n_a. However, compared with the smoother performance curve of DeepSeek-V2-Lite, DeepSeek-V3 exhibits greater instability across different reduction strategies. This instability may stem from the inherent properties of the sigmoid function, where expert weights tend to polarize toward 0 or 1. In contrast, in softmax-based models, the weights of lower-ranked experts are significantly smaller than that of the top-ranked expert. One possible explanation is that sigmoid routing does not enforce a normalized competition across experts, which can make the retained experts contribute more uniformly; consequently, removing any high-weight expert may be more damaging.

Across the evaluated layouts, the best-performing pattern differs between DeepSeek-V2-Lite and DeepSeek-V3: descending reductions work best for DeepSeek-V2-Lite, whereas ascending reductions work best for DeepSeek-V3. This contrast suggests that static skipping in fine-grained MoEs does leave usable quality headroom, but the best way to allocate that headroom is model-specific rather than universal. It therefore motivates moving beyond pre-defined layouts toward mechanisms that can use token-level runtime information. In this sense, layer-wise schedules remain a coarse proxy, since they allocate the same budget to all tokens that traverse a layer.

5.5 Adaptive Evaluation with ASET

We next examine whether token-adaptive control can provide finer-grained control over the trade-off between average activation and task quality within the same training-free, within-budget expert-skipping setting. Rather than choosing among a small number of discrete static layouts, ASET is intended to trace a continuous family of operating points under the same deployment constraint. We use OLMoE as the primary testbed for this adaptive evaluation, since its router produces a normalized distribution that naturally supports entropy-based calibration and provides a cleaner setting for isolating the effect of adap-

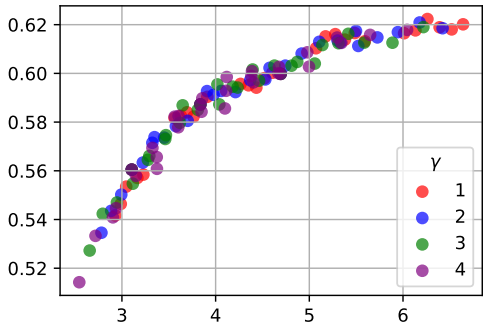

Figure 6: ASET speed–quality trade-off. Each point corresponds to a fixed $(\theta_{\min}, \theta_{\max}, \gamma)$ and reports average activated experts $\bar{k}$ (x-axis) versus average task score (y-axis) over all tasks.

tive activation. We sweep ASET hyperparameters $(\theta_{\min}, \theta_{\max}, \gamma)$ and report, for each configuration, the resulting average number of activated experts $\bar{k}$ and the average task score over our benchmark suite.

Figure 6 visualizes the Pareto frontier induced by ASET. As $\bar{k}$ increases, the average score improves monotonically with diminishing returns, indicating that ASET provides a continuum of operating points rather than a small set of fixed activation levels. This behavior is useful operationally because it allows the activation budget to be adjusted more smoothly than with a small menu of pre-defined static schedules. Moreover, points with different curvature parameters γ largely lie on the same frontier, suggesting that the resulting trade-off is not highly sensitive to γ within the tested range. In practice, $\bar{k}$ is controlled mainly through $(\theta_{\min}, \theta_{\max})$, while γ plays a secondary shaping role in how aggressively the threshold responds to ambiguity. Overall, this experiment shows that token-adaptive control can induce a stable Pareto trade-off between average activation and task quality on OLMoE within the original expert-skipping budget.

6 Conclusion

We revisit inference-time expert activation for fine-grained MoEs. Despite nominal FLOPs comparable to dense FFNs, efficient execution remains challenging in practice. By characterizing uniform skipping and searched static layer-wise Top-k schedules, we show that expert skipping is a practical throughput lever, delivering 10–78% gains across representative fine-grained MoEs with little degradation, including over 10% on DeepSeek-V3 without measurable loss.

We also find fixed-budget activation brittle: preferred static patterns vary across models and routing mechanisms, even reversing between DeepSeek-V2-Lite and DeepSeek-V3. We therefore propose Adaptive Skipping with Entropy-Penalized Thresholding (ASET), a training-free policy that adapts per-token expert activation using router confidence and entropy while remaining within the original activation budget. On the OLMoE testbed, ASET yields a Pareto frontier between average activation and task quality. Overall, our results show that expert skipping is a practical lever for faster fine-grained MoE inference, with ASET illustrating the value of adaptive activation when fixed schedules are too rigid.

Acknowledgements

This work was supported by the National Science and Technology Major Project (No. 2023ZD0121502), the National Social Science Foundation of China (No. 24&ZD186), and the National Natural Science Foundation of China (No. 62306216). Hao Huang's contribution was supported by the Science and Technology Program of XPCC (No. 2025AA017).

Limitations

We focus on *fine-grained* MoE, where inference-time compute allocation is a central design dimension; coarse-grained MoE offers much less budget headroom, and dense models do not expose an analogous activation control. We use additional MoE families to show that static layer-wise schedules are not portable across routing mechanisms. For adaptive evaluation, we prioritize OLMoE, whose normalized router outputs provide a cleaner setting for entropy-based calibration. Evidence for ASET is therefore concentrated on OLMoE rather than all routing families, and we do not claim universal static schedules.

We report throughput as the primary system metric under controlled decoding. Our evaluation therefore does not directly characterize deployment-level latency under realistic traffic, batching, or queueing conditions, or expert-parallel communication effects. Training-aware routing modification and finetuning-based adaptation are complementary but fall outside our training-free scope.

References

Luisa Bentivogli, Bernardo Magnini, Ido Dagan, Hoa Trang Dang, and Danilo Giampiccolo. 2009. The fifth PASCAL recognizing textual entailment challenge. In *Proceedings of the Second Text Analysis Conference, TAC 2009, Gaithersburg, Maryland, USA, November 16-17, 2009*. NIST.

I-Chun Chen, Hsu-Shen Liu, Wei-Fang Sun, Chen-Hao Chao, Yen-Chang Hsu, and Chun-Yi Lee. 2025. Retraining-free merging of sparse mixture-of-experts via hierarchical clustering.

Tianyu Chen, Shaohan Huang, Yuan Xie, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. 2022. Task-specific expert pruning for sparse mixture-of-experts. *CoRR*, abs/2206.00277.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, and 48 others. 2023. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457.

Damai Dai, Chengqi Deng, Chenggang Zhao, R.x. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y.k. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1280–1297, Bangkok, Thailand. Association for Computational Linguistics.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23:120:1–120:39.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Dan Roberts. 2025. The unreasonable ineffectiveness of the deeper layers. In *The Thirteenth International Conference on Learning Representations*.

Junhui He, Shangyu Wu, Weidong Wen, Chun Jason Xue, and Qingan Li. 2024. CHESS: Optimizing LLM inference via channel-wise thresholding and selective sparsification. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18658–18668, Miami, Florida, USA. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pages 611–626, New York, NY, USA. Association for Computing Machinery.

Jaeseong Lee, Seung-won Hwang, Aurick Qiao, Daniel F. Campos, Zhewei Yao, and Yuxiong He. 2024. STUN: structured-then-unstructured pruning for scalable moe pruning. *CoRR*, abs/2409.06211.

Pingzhi Li, Zhenyu Zhang, Prateek Yadav, Yi-Lin Sung, Yu Cheng, Mohit Bansal, and Tianlong Chen. 2024. Merge, then compress: Demystify efficient smoe with hints from its routing policy. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. 2024. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6159–6172, Bangkok, Thailand. Association for Computational Linguistics.

Shi Luohe, Zuchao Li, Lefei Zhang, Baoyuan Qi, Liu Guoming, and Hai Zhao. 2025. KV-latent: Dimensional-level KV cache reduction with frequency-aware rotary positional embedding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1550, Vienna, Austria. Association for Computational Linguistics.

Ziyang Ma, Zuchao Li, Lefei Zhang, Gui-Song Xia, Bo Du, Liangpei Zhang, and Dacheng Tao. 2026. Phase transitions in large language model compression. *npj Artificial Intelligence*, 2(1):21.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium. Association for Computational Linguistics.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, and 5 others. 2024. Olmoe: Open mixture-of-experts language models. *Preprint*, arXiv:2409.02060.

Alexandre Muzio, Alex Sun, and Churan He. 2024. Seer-moe: Sparse expert efficiency through regularization for mixture-of-experts. *CoRR*, abs/2404.05089.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106.

Noam Shazeer. 2020. GLU variants improve transformer. *CoRR*, abs/2002.05202.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Luohe Shi, Zuchao Li, Lefei Zhang, Baoyuan Qi, Guoming Liu, and Hai Zhao. 2026. Scaling LLM speculative decoding: Non-autoregressive forecasting in large-batch scenarios. In *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pages 32947–32955. AAAI Press.

Zhibin Wang, Shipeng Li, Yuhang Zhou, Xue Li, Rong Gu, Nguyen Cam-Tu, Chen Tian, and Sheng Zhong. 2024. Revisiting SLO and goodput metrics in LLM serving. *CoRR*, abs/2410.14257.

Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Yanyue Xie, Zhi Zhang, Ding Zhou, Cong Xie, Ziang Song, Xin Liu, Yanzhi Wang, Xue Lin, and An Xu. 2024. Moe-pruner: Pruning mixture-of-experts large language model using the hints from its router. *CoRR*, abs/2410.12013.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024a. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Cheng Yang, Yang Sui, Jinqi Xiao, Lingyi Huang, Yu Gong, Yuanlin Duan, Wenqi Jia, Miao Yin, Yu Cheng, and Bo Yuan. 2024b. Moe-i^2: Compressing mixture of experts models through inter-expert pruning and intra-expert low-rank decomposition. *CoRR*, abs/2411.01016.

Haoqi Yang, Yao Yao, Zuchao Li, Baoyuan Qi, Liu Guoming, and Hai Zhao. 2025. XQuant: Achieving ultra-low bit KV cache quantization with cross-layer compression. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 9785–9800, Suzhou, China. Association for Computational Linguistics.

Lu Ye, Ze Tao, Yong Huang, and Yang Li. 2024. ChunkAttention: Efficient self-attention with prefix-aware KV cache and two-phase partition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11608–11620, Bangkok, Thailand. Association for Computational Linguistics.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA. USENIX Association.

Zihao Zeng, Yibo Miao, Hongcheng Gao, Hao Zhang, and Zhijie Deng. 2024. AdaMoE: Token-adaptive routing with null experts for mixture-of-experts language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6223–6235, Miami, Florida, USA. Association for Computational Linguistics.

A Pre-Inference Expert Pruning

Beyond reducing the number of activated experts during inference, we also explore another possibility introduced by fine-grained MoE models, determining which experts to discard before inference begins. In other words, this corresponds to reducing the total number of experts (n_e). This approach has already been partially investigated, primarily in the context of fine-grained models, where expert pruning is typically guided by information-based metrics or performance comparisons after expert removal.

However, we focus on two additional key questions: 1. To what extent can these methods accelerate inference? 2. Are these methods still effective for fine-grained MoE models? While reducing the total number of experts does not decrease the computational workload, it can increase computational intensity, which could theoretically lead to a net positive effect on inference speed. Furthermore, fine-grained MoE models introduce a unique challenge for global expert reduction, fully randomized initialization, meaning that experts do not exhibit any similarities, making effective global pruning significantly more complex.

A.1 Efficiency

We conducted experiments on DeepSeek-V2-Lite with varying n_e under a concurrent request setting of 512. The results of our evaluation are presented in Figure 7.

We observed that the acceleration ratio was also significant at lower throughput levels, with up to 2.3Œ speedup. This is because when the number of experts is reduced, the computational intensity per expert increases rapidly, leading to a substantial increase in inference speed even with unchanged FLOPs. However, we noticed that at a concurrency level of 192, the throughput after expert reduction decreased.

A.2 Performance

We evaluated the performance of different selection strategies under varying n_e values, as presented in Table 1. We considered several expert reduction strategies, including random removal, structured removal (odd indices, even indices, lower-half indices, and upper-half indices, which are theoretically equivalent to random selection), and the soft-count and hard-count methods proposed by Muzio et al., which estimate expert importance by their popularity before selecting the most important experts. In some cases, certain methods lost capability on specific test sets and performed at a level comparable to random guessing; we mark such cases in *italics*.

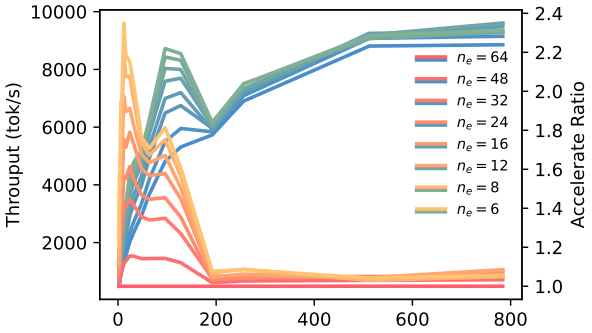

Figure 7: Throughput and speedup of expert pruning in DeepSeek-V2-Lite.

Our results indicate that the soft count method consistently outperformed the others. When reducing the number of experts by 25%, even the best method exhibited some performance degradation, whereas random selection resulted in significant accuracy loss. With a 50% reduction in experts, even the best method experienced a 15% drop in performance, while randomly selected experts completely lost language capabilities. Finally, when only 25% of the experts were retained, random selection still resulted in total failure, and only the best method was able to maintain some capability, while even the slightly inferior hard count method almost entirely lost its effectiveness.

An interesting observation from our experiments is that structured removal outperformed purely random selection. More specifically, when retaining experts with even indices or those from the latter half, performance was close to that of the best method. In contrast, the other two structured methods led to near-total language capability loss. This suggests the existence of particularly critical experts, which happen to be those with even indices and higher-numbered designations. In other words, despite the load-balancing mechanisms applied during training, there remains a significant disparity in expert importance.

A.3 Combination

We integrated both methods to evaluate the extent of efficiency improvement, as illustrated in Figure 8 and Figure 9. It's evident that for smaller batch sizes, reducing the total number of experts does not have as significant an impact as reducing

Method	n_e	ARC-C	ARC-E	BoolQ	OBQA	RTE	WinoGrande	Avg
Baseline	64	52.9	80.6	83.1	35.8	73.3	71.4	66.0
Random	16	*20.8*	27.2	60.8	*14.8*	*50.5*	*50.5*	*37.4*
	32	*19.2*	31.1	59.4	*15.2*	*50.5*	*51.2*	*37.7*
	48	43.7	75.0	81.2	31.2	65.7	67.5	60.7
Odd	32	*22.2*	30.9	65.9	*19.8*	57.0	54.7	41.7
Even	32	32.1	62.2	69.4	*23.4*	66.1	63.7	52.8
First half	32	*21.4*	30.8	62.5	*21.6*	59.9	*52.7*	41.5
Last half	32	32.6	60.1	75.3	*23.6*	71.5	65.4	54.7
Activate Count	16	*21.8*	31.4	62.0	*14.8*	57.0	*50.7*	39.6
	32	40.8	68.3	75.0	29.2	54.9	70.2	56.3
	48	44.7	75.8	79.4	33.8	66.4	72.6	62.1
Soft Count	16	28.7	57.3	62.2	*22.4*	55.6	60.9	47.8
	32	39.7	71.2	76.1	31.4	58.5	70.2	57.8
	48	46.8	76.3	80.0	33.8	76.5	72.1	64.2

Table 1: Performance of different expert pruning method.

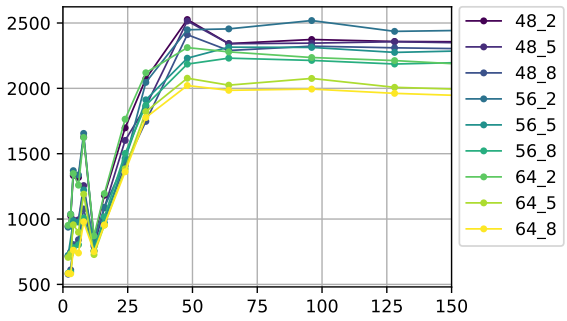

Figure 8: Throughput comparison of combined optimization strategies under varying concurrency.

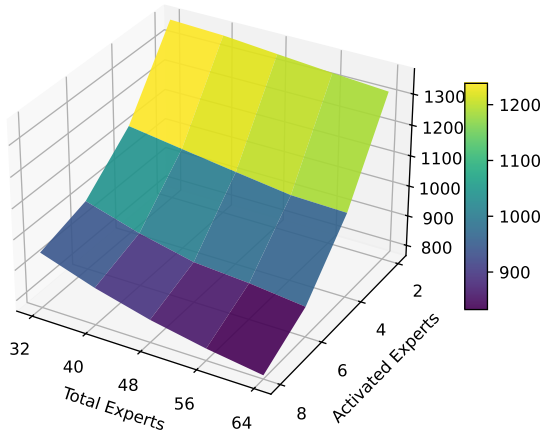

Figure 9: Throughput landscape resulting from the combination of expert pruning and expert skipping at a fixed concurrency of 6.

the number of activated experts. The advantage of reducing the total number of experts only becomes apparent when the batch size is sufficiently large, though its intensity is only approximate to that of reducing the number of activated experts. Specifically, as depicted in Figure 9 for a concurrency of 6, combining both methods can boost throughput by as much as 78%.

B Sensitivity of Throughput to Input/Output Length Configuration

We evaluate throughput at concurrency 512 while fixing the total length to 2048 tokens and varying the input/output split. Table 2 reports the full sweep for DeepSeek-V2-Lite, and Table 3 reports selected DeepSeek-V3 measurements. In both models, throughput increases as the input-token share grows, reflecting the greater parallelism of prefill relative to autoregressive decoding. This sensitivity motivates the fixed 1024/1024 setting used throughout our efficiency evaluation, which is intended for controlled comparison rather than to match any particular workload distribution.

Input token	256	512	768	1024	1280	1536	1792
Output token	1792	1536	1280	1024	768	512	256
Throughput	6368	7106	8224	9484	10419	11584	13040

Table 2: Influence of IO token counts on throughput for DeepSeek-V2-Lite.

Input token	1024	1024
Output token	1024	8
Throughput	2636	8487

Table 3: Influence of IO token counts on throughput for DeepSeek-V3.

C Implementation Details for Section 3.2

C.1 Hardware

Please reference Table 4.

Item	Type	Quantity
CPU	Intel Xeon Silver 4314 CPU @ 2.40GHz	24
GPU	NVIDIA Tesla A800 80G PCI-e	1
Memory	16GB ECC DDR4@2666MHz	15

Table 4: Hardware Information

C.2 Software

We use PyTorch with `torch.compile` as the base framework. For GLU, we use a SwiGLU implementation, and for MoE we use the `MixtralModel` implementation from Transformers (Wolf et al., 2020).

D Implementation Details for Section 4

D.1 DeepSeek-V2-Lite

D.1.1 Hardware

Please reference Table 5.

Item	Type	Quantity
CPU	Intel Xeon Silver 4314 CPU @ 2.40GHz	24
GPU	NVIDIA Tesla A800 80G PCI-e	2
Memory	16GB ECC DDR4@2666MHz	15

Table 5: Hardware Information

D.1.2 Software

We utilize sglang build v0.4.4 post 1 (commit `ad4e58bf67ec833ff4d036af5129ec6e1633efc4`) as the backend and sglang.bench for profiling.

D.2 DeepSeek-V3

D.2.1 Hardware

Please reference Table 6.

D.2.2 Software

We utilize sglang build v0.4.4 post 1 (commit `ad4e58bf67ec833ff4d036af5129ec6e1633efc4`) as the backend and sglang.bench for profiling.

Item	Type	Quantity
CPU	Intel Xeon Platinum 8558 CPU @ 2.10GHz	48x2
GPU	NVIDIA Tesla H200 141G SXM5	8
Memory	64GB ECC DDR4@2666MHz	32

Table 6: Hardware Information

E Detailed Results for Section 4 and Appendix A

Reference Table 7 for Figure 4a. Reference Table 8 for Figure 4b. Reference Table 9 for Figure 5a. Reference Table 10 for Figure 5b. Reference Table 11 for Figure 7.

Concurrency	$n_a =$ 6	5	4	3	2
2	479	511	544	583	631
4	729	774	838	974	1099
8	1041	1122	1234	1380	1581
16	1519	1608	1738	1932	2254
32	2345	2412	2529	2716	3069
48	2968	3017	3111	3258	3572
64	3755	3801	3896	4042	4357
96	5020	5087	5172	5279	5608
128	5591	5660	5812	5960	6126
192	5948	6104	6069	6227	6461
256	7263	7385	7430	7501	7846
384	9052	9218	9369	9551	10137
512	9379	9783	9950	10102	10954
768	9453	9694	10043	10249	10968

Table 7: Detailed result of Figure 4a.

Concurrency	$n_a =$ 8	6	4	3
2	163	164	167	170
4	300	308	312	323
8	526	537	554	575
16	769	793	820	979
32	1204	1331	1401	1705
64	1666	1751	1851	1900
128	2751	2806	2985	3100

Table 8: Detailed result of Figure 4b.

(b, h, e, p)	ARC-C	ARC-E	BoolQ	OBQA	RTE	WinoGrande	Avg
(2, 2, 2, 1)	43.1	73.7	80.3	30.0	69.7	65.8	60.4
(2, 2, 2, 6)	43.6	73.9	80.3	30.0	69.7	66.9	60.7
(2, 2, 2, 11)	43.5	73.9	80.2	30.0	69.3	65.6	60.4
(2, 2, 2, 16)	43.5	74.0	80.3	30.0	69.7	65.0	60.4
(2, 2, 2, 21)	42.6	73.6	80.2	30.0	69.3	65.6	60.2
(2, 2, 2, 26)	43.3	73.4	80.2	30.4	69.7	65.5	60.4
(2, 2, 4, 1)	45.1	76.4	80.6	33.0	72.6	68.2	62.6
(2, 2, 4, 6)	45.1	76.0	79.7	31.0	71.8	66.3	61.6
(2, 2, 4, 11)	43.7	75.4	79.9	31.4	70.8	65.8	61.2
(2, 2, 4, 16)	43.5	74.8	80.2	29.6	70.8	65.4	60.7
(2, 2, 4, 21)	42.2	74.7	80.2	30.2	70.0	66.4	60.6
(2, 2, 4, 26)	42.8	73.5	80.5	30.2	69.7	64.6	60.2
(2, 2, 6, 1)	47.4	77.8	80.9	34.0	76.2	68.0	64.0
(2, 2, 6, 6)	46.5	77.0	80.3	32.2	70.8	67.1	62.3
(2, 2, 6, 11)	44.7	76.3	80.3	32.4	72.2	65.9	62.0
(2, 2, 6, 16)	43.4	75.0	80.0	30.6	71.1	66.2	61.1
(2, 2, 6, 21)	43.0	74.7	80.1	31.0	72.2	65.4	61.1
(2, 2, 6, 26)	43.5	73.8	80.2	29.8	71.1	65.7	60.7
(2, 4, 2, 1)	49.4	77.9	82.4	33.2	72.2	69.6	64.1
(2, 4, 2, 6)	48.3	77.7	82.9	32.8	71.8	69.9	63.9
(2, 4, 2, 11)	47.7	78.2	82.2	32.4	72.9	70.1	63.9
(2, 4, 2, 16)	48.2	77.2	81.5	33.6	73.3	68.7	63.8
(2, 4, 2, 21)	46.6	77.5	81.9	33.2	73.3	68.5	63.5
(2, 4, 2, 26)	46.4	76.9	80.7	33.0	72.6	66.7	62.7
(2, 4, 4, 1)	49.6	78.2	83.1	33.4	74.0	70.7	64.8
(2, 4, 4, 6)	48.0	78.5	82.4	33.4	74.7	70.5	64.6
(2, 4, 4, 11)	48.5	78.6	81.8	32.4	74.0	70.2	64.2
(2, 4, 4, 16)	47.6	77.5	81.2	34.8	75.1	68.8	64.2
(2, 4, 4, 21)	46.8	77.9	81.4	34.0	73.3	68.6	63.7
(2, 4, 4, 26)	45.6	76.6	80.7	33.4	71.1	67.3	62.4
(2, 4, 6, 1)	50.5	78.9	82.2	34.4	74.0	69.9	65.0
(2, 4, 6, 6)	48.4	78.5	82.5	32.6	74.7	70.9	64.6
(2, 4, 6, 11)	48.1	78.7	81.4	33.0	73.6	69.4	64.0
(2, 4, 6, 16)	48.0	77.7	81.2	33.6	74.4	69.8	64.1
(2, 4, 6, 21)	46.2	77.3	81.2	33.6	72.9	67.9	63.2
(2, 4, 6, 26)	46.2	76.6	80.9	33.2	71.8	66.6	62.6
(2, 6, 2, 1)	50.1	79.1	83.5	32.8	71.8	71.1	64.8
(2, 6, 2, 6)	48.5	78.5	82.8	34.4	74.0	69.9	64.7
(2, 6, 2, 11)	50.1	78.5	82.6	33.8	73.3	70.7	64.8
(2, 6, 2, 16)	48.7	78.4	82.8	33.6	73.3	70.3	64.5
(2, 6, 2, 21)	48.3	78.9	82.1	33.8	74.7	71.0	64.8
(2, 6, 2, 26)	49.4	77.7	81.3	34.8	73.6	68.3	64.2
(2, 6, 4, 1)	50.5	79.6	83.1	33.8	72.9	71.7	65.3
(2, 6, 4, 6)	50.3	79.3	82.8	34.4	73.6	71.6	65.3
(2, 6, 4, 11)	49.2	78.4	81.9	33.2	72.6	70.9	64.4
(2, 6, 4, 16)	48.4	79.1	82.4	33.6	75.1	70.4	64.8
(2, 6, 4, 21)	49.0	78.5	82.0	33.2	74.7	70.6	64.7
(2, 6, 4, 26)	48.9	77.8	81.2	34.4	72.6	68.2	63.8
(2, 6, 6, 1)	50.9	80.1	83.0	35.6	72.9	71.9	65.7
(2, 6, 6, 6)	51.0	79.5	82.4	34.4	74.0	72.0	65.5
(2, 6, 6, 11)	49.7	78.7	82.2	33.4	72.9	71.5	64.7
(2, 6, 6, 16)	48.8	78.7	82.1	34.0	75.1	71.1	65.0
(2, 6, 6, 21)	47.9	78.7	81.9	33.2	74.0	70.6	64.4
(2, 6, 6, 26)	48.6	77.5	81.3	34.2	75.1	68.0	64.1

(b, h, e, p)	ARC-C	ARC-E	BoolQ	OBQA	RTE	WinoGrande	Avg
(4, 2, 2, 1)	44.1	74.5	80.7	30.6	70.4	67.2	61.3
(4, 2, 2, 6)	45.5	75.6	81.0	31.4	71.8	67.7	62.2
(4, 2, 2, 11)	47.4	76.5	81.9	31.4	72.6	68.2	63.0
(4, 2, 2, 16)	48.4	77.1	82.2	32.6	73.3	67.9	63.6
(4, 2, 2, 21)	49.7	77.2	82.6	32.6	72.6	68.0	63.8
(4, 2, 2, 26)	49.1	77.7	82.5	31.8	72.9	69.3	63.9
(4, 2, 4, 1)	47.5	77.9	81.9	32.0	74.7	69.6	63.9
(4, 2, 4, 6)	47.7	76.4	80.7	30.6	73.3	68.0	62.8
(4, 2, 4, 11)	47.9	77.5	82.1	31.6	74.0	69.1	63.7
(4, 2, 4, 16)	47.4	77.5	81.8	32.0	75.1	69.5	63.9
(4, 2, 4, 21)	49.7	77.9	82.3	32.6	74.0	68.3	64.1
(4, 2, 4, 26)	49.0	78.4	82.3	32.2	72.6	69.3	63.9
(4, 2, 6, 1)	48.4	77.3	81.3	34.0	71.8	70.0	63.8
(4, 2, 6, 6)	48.5	77.6	81.1	32.6	73.6	68.8	63.7
(4, 2, 6, 11)	48.4	78.4	82.0	31.8	74.0	69.1	63.9
(4, 2, 6, 16)	47.9	78.1	82.1	32.8	74.0	69.3	64.0
(4, 2, 6, 21)	48.9	77.9	82.6	32.2	73.6	69.6	64.1
(4, 2, 6, 26)	49.2	78.0	82.6	32.0	72.6	68.4	63.8
(4, 4, 2, 1)	48.6	77.5	82.8	34.2	70.8	69.5	63.9
(4, 4, 2, 6)	50.1	78.4	82.8	32.8	72.2	70.5	64.5
(4, 4, 2, 11)	49.7	78.5	82.8	34.2	72.2	70.6	64.6
(4, 4, 2, 16)	50.2	77.9	82.7	34.2	73.3	69.9	64.7
(4, 4, 2, 21)	50.9	78.2	82.8	34.4	72.9	70.6	65.0
(4, 4, 2, 26)	50.5	79.0	82.8	33.6	73.3	70.4	64.9
(4, 4, 4, 1)	50.5	79.1	82.7	34.8	73.3	70.7	65.2
(4, 4, 4, 6)	50.7	79.4	82.8	34.8	73.3	70.3	65.2
(4, 4, 4, 11)	51.0	79.0	82.7	34.4	73.6	70.1	65.1
(4, 4, 4, 16)	51.3	79.0	82.7	34.8	73.3	70.3	65.2
(4, 4, 4, 21)	50.3	78.9	82.6	34.8	74.0	70.8	65.2
(4, 4, 4, 26)	50.9	79.0	82.7	35.2	73.6	70.5	65.3
(4, 4, 6, 1)	50.2	79.5	83.1	33.8	74.4	71.1	65.4
(4, 4, 6, 6)	51.6	79.0	82.9	33.6	73.6	70.9	65.3
(4, 4, 6, 11)	51.4	79.0	82.8	34.6	74.4	70.2	65.4
(4, 4, 6, 16)	51.2	79.0	82.9	33.8	73.3	70.2	65.1
(4, 4, 6, 21)	50.8	79.0	82.6	34.0	74.0	70.6	65.2
(4, 4, 6, 26)	50.4	78.9	82.7	34.4	73.3	70.6	65.1
(4, 6, 2, 1)	51.7	79.5	83.5	34.6	72.9	70.2	65.4
(4, 6, 2, 6)	50.9	79.5	83.5	35.0	72.6	71.9	65.6
(4, 6, 2, 11)	52.1	79.5	83.3	33.6	73.6	70.7	65.5
(4, 6, 2, 16)	51.8	78.9	83.4	33.8	73.3	70.2	65.2
(4, 6, 2, 21)	51.8	79.5	83.2	35.0	74.0	71.3	65.8
(4, 6, 2, 26)	50.8	79.5	83.1	33.4	73.3	71.3	65.2
(4, 6, 4, 1)	52.1	80.1	83.5	35.0	72.6	71.8	65.9
(4, 6, 4, 6)	52.5	80.0	83.1	35.0	74.0	71.8	66.1
(4, 6, 4, 11)	52.2	80.4	83.2	33.8	74.0	71.7	65.9
(4, 6, 4, 16)	52.1	79.5	83.1	34.0	74.0	70.7	65.6
(4, 6, 4, 21)	51.5	79.4	83.3	34.4	73.6	71.3	65.6
(4, 6, 4, 26)	50.7	79.6	83.0	33.6	73.6	71.3	65.3
(4, 6, 6, 1)	52.7	80.2	83.0	34.6	73.3	71.7	65.9
(4, 6, 6, 6)	52.2	80.1	83.2	34.8	73.3	72.7	66.1
(4, 6, 6, 11)	52.9	79.9	83.0	34.2	73.6	72.5	66.0
(4, 6, 6, 16)	51.9	79.5	83.1	35.2	74.0	71.6	65.9
(4, 6, 6, 21)	51.1	79.7	83.2	34.6	74.0	71.7	65.7
(4, 6, 6, 26)	51.2	79.7	83.3	33.8	74.0	71.3	65.6

(b, h, e, p)	ARC-C	ARC-E	BoolQ	OBQA	RTE	WinoGrande	Avg
(6, 2, 2, 1)	44.8	75.1	80.7	31.4	70.0	66.0	61.3
(6, 2, 2, 6)	46.3	75.8	81.7	32.4	72.2	68.0	62.7
(6, 2, 2, 11)	47.5	77.2	82.4	32.8	72.6	68.8	63.6
(6, 2, 2, 16)	49.5	78.4	83.0	34.0	72.6	70.2	64.6
(6, 2, 2, 21)	50.7	78.8	83.9	33.8	72.6	69.3	64.8
(6, 2, 2, 26)	50.8	79.3	83.5	34.4	71.5	71.6	65.2
(6, 2, 4, 1)	47.8	77.2	80.6	32.2	70.8	67.9	62.7
(6, 2, 4, 6)	48.0	77.3	81.3	32.4	73.6	67.9	63.4
(6, 2, 4, 11)	48.3	78.2	82.6	32.6	73.3	69.3	64.1
(6, 2, 4, 16)	48.5	78.5	83.0	34.0	74.7	70.5	64.9
(6, 2, 4, 21)	50.5	79.1	83.8	33.6	72.6	70.2	65.0
(6, 2, 4, 26)	51.5	79.4	83.7	35.4	72.2	71.3	65.6
(6, 2, 6, 1)	48.9	77.9	81.2	33.4	72.2	68.8	63.7
(6, 2, 6, 6)	49.0	77.9	81.4	32.8	74.4	69.5	64.2
(6, 2, 6, 11)	50.0	78.9	82.4	32.8	73.6	70.4	64.7
(6, 2, 6, 16)	50.2	79.3	83.1	34.4	74.7	71.2	65.5
(6, 2, 6, 21)	50.9	79.4	83.6	34.0	73.3	69.5	65.1
(6, 2, 6, 26)	51.0	79.7	83.7	35.2	71.1	71.3	65.3
(6, 4, 2, 1)	49.5	77.8	82.8	33.8	72.6	69.8	64.4
(6, 4, 2, 6)	50.8	79.1	83.5	33.6	74.0	70.2	65.2
(6, 4, 2, 11)	51.0	79.2	83.8	34.8	72.2	70.6	65.3
(6, 4, 2, 16)	51.7	80.0	83.6	33.6	72.6	70.1	65.3
(6, 4, 2, 21)	52.2	80.1	83.4	35.4	72.9	70.4	65.7
(6, 4, 2, 26)	52.6	80.3	83.5	35.6	72.6	71.7	66.1
(6, 4, 4, 1)	51.2	79.6	83.3	34.6	74.7	70.2	65.6
(6, 4, 4, 6)	50.7	79.9	83.2	35.0	74.0	71.2	65.7
(6, 4, 4, 11)	51.9	80.1	83.7	34.6	73.3	71.0	65.8
(6, 4, 4, 16)	51.9	80.5	83.5	34.4	72.9	70.6	65.6
(6, 4, 4, 21)	52.2	80.1	83.4	35.0	72.6	71.3	65.8
(6, 4, 4, 26)	52.4	80.2	83.4	35.6	72.6	71.6	65.9
(6, 4, 6, 1)	51.6	79.3	83.3	34.4	74.4	70.8	65.6
(6, 4, 6, 6)	51.4	79.4	83.3	34.0	74.0	72.3	65.7
(6, 4, 6, 11)	52.4	79.7	83.7	33.8	72.9	70.2	65.4
(6, 4, 6, 16)	51.5	80.3	83.6	34.4	73.3	71.1	65.7
(6, 4, 6, 21)	51.6	80.2	83.3	35.0	72.2	71.2	65.6
(6, 4, 6, 26)	52.1	80.2	83.4	35.8	72.6	71.9	66.0
(6, 6, 2, 1)	50.3	80.1	83.6	34.4	71.8	70.8	65.2
(6, 6, 2, 6)	52.0	80.4	83.5	35.2	72.6	70.5	65.7
(6, 6, 2, 11)	51.5	80.2	83.5	35.0	71.8	71.0	65.5
(6, 6, 2, 16)	52.3	80.0	83.6	36.2	71.8	71.3	65.9
(6, 6, 2, 21)	52.6	80.3	83.3	36.4	72.2	71.3	66.0
(6, 6, 2, 26)	52.7	80.5	83.2	35.2	72.6	71.3	65.9
(6, 6, 4, 1)	52.0	80.4	83.6	35.4	72.6	71.4	65.9
(6, 6, 4, 6)	52.3	80.4	83.3	36.0	72.2	71.7	66.0
(6, 6, 4, 11)	52.6	80.6	83.2	35.6	72.2	70.9	65.8
(6, 6, 4, 16)	52.7	80.4	83.3	35.2	72.6	72.0	66.0
(6, 6, 4, 21)	53.2	80.3	83.1	36.0	72.6	71.9	66.2
(6, 6, 4, 26)	53.2	80.3	83.1	35.6	72.6	70.6	65.9
(6, 6, 6, 1)	53.3	80.3	83.3	34.8	72.6	70.9	65.9
(6, 6, 6, 6)	52.8	80.5	83.2	35.6	72.9	70.6	65.9
(6, 6, 6, 11)	52.8	80.1	83.1	35.8	72.9	71.4	66.0
(6, 6, 6, 16)	52.9	80.0	83.1	35.8	73.3	71.2	66.0
(6, 6, 6, 21)	52.9	80.6	83.0	35.8	72.6	71.0	66.0
(6, 6, 6, 26)	52.9	80.5	83.1	35.4	72.9	71.0	66.0

Table 9: Detailed result of Figure 5a.

(b, h, e, p)	ARC-C	ARC-E	BoolQ	OBQA	RTE	WinoGrande	Avg
(2, 2, 2, 61)	53.2	78.9	69.9	33.2	65.3	64.5	60.8
(2, 4, 4, 61)	58.1	84.4	78.6	36.4	69.7	73.6	66.8
(2, 6, 6, 61)	63.1	85.4	83.8	39.2	71.1	76.2	69.8
(2, 8, 8, 61)	65.4	87.0	86.9	37.4	73.6	78.2	71.4
(4, 2, 2, 61)	57.9	82.5	78.9	35.0	69.3	72.5	66.0
(4, 4, 4, 61)	64.1	85.5	85.6	38.4	71.8	78.2	70.6
(4, 6, 6, 61)	65.5	86.8	87.3	38.6	73.6	78.8	71.8
(4, 8, 8, 61)	65.3	87.6	88.0	38.0	74.0	80.2	72.2
(6, 2, 2, 61)	61.9	85.5	84.1	36.2	73.3	75.1	69.3
(6, 4, 4, 61)	64.2	87.1	87.4	37.8	73.6	78.4	71.4
(6, 6, 6, 61)	65.8	87.1	88.5	39.8	76.2	79.9	72.9
(6, 8, 8, 61)	66.0	87.9	88.6	39.2	77.6	80.6	73.3
(8, 2, 2, 61)	62.8	85.4	87.3	37.4	74.0	77.2	70.7
(8, 4, 4, 61)	65.2	87.3	88.4	38.0	75.8	79.0	72.3
(8, 6, 6, 61)	66.5	87.8	89.0	39.6	76.2	79.4	73.1
(8, 8, 8, 61)	65.5	88.0	89.2	39.6	75.5	80.4	73.0
(2, 5, 2, 10)	61.3	85.4	83.5	34.6	74.0	74.5	68.9
(2, 5, 2, 30)	62.8	85.5	84.7	37.4	73.3	75.4	69.8
(2, 5, 2, 50)	62.4	85.0	81.9	36.4	70.8	73.8	68.4
(2, 5, 8, 10)	66.0	87.8	87.9	39.4	77.3	79.0	72.9
(2, 5, 8, 30)	65.9	87.3	86.9	38.2	73.6	78.7	71.8
(2, 5, 8, 50)	62.0	85.7	83.5	37.6	71.1	76.6	69.4
(2, 8, 2, 10)	64.7	85.9	87.0	36.8	73.6	78.1	71.0
(2, 8, 2, 30)	66.0	87.0	87.5	38.0	76.5	77.9	72.2
(2, 8, 2, 50)	64.2	86.1	86.1	37.4	72.6	78.5	70.8
(2, 8, 5, 10)	65.5	87.5	89.2	38.6	75.8	79.6	72.7
(2, 8, 5, 30)	65.8	87.4	88.5	37.6	74.4	79.2	72.1
(2, 8, 5, 50)	64.5	87.0	87.3	37.8	74.4	79.6	71.8
(5, 2, 5, 10)	59.9	85.0	80.5	34.8	69.0	73.4	67.1
(5, 2, 5, 30)	58.4	83.6	79.7	34.6	69.3	72.2	66.3
(5, 2, 5, 50)	60.7	85.4	82.6	36.0	73.6	73.4	68.6
(5, 2, 8, 10)	62.6	86.4	84.9	38.6	67.1	76.9	69.4
(5, 2, 8, 30)	60.0	84.7	81.2	36.8	66.8	73.6	67.2
(5, 2, 8, 50)	60.3	85.7	83.3	35.4	72.6	75.1	68.7
(5, 8, 2, 10)	64.0	85.7	87.0	38.4	74.7	79.1	71.5
(5, 8, 2, 30)	64.4	86.7	88.5	37.0	75.5	78.4	71.8
(5, 8, 2, 50)	64.9	87.2	89.0	37.0	75.5	79.0	72.1
(5, 8, 5, 10)	65.9	87.5	89.1	38.8	77.6	80.0	73.2
(5, 8, 5, 30)	65.5	87.7	89.1	39.6	75.8	79.9	72.9
(5, 8, 5, 50)	65.4	87.6	89.1	38.4	75.1	80.5	72.7
(8, 2, 5, 10)	61.6	84.8	80.4	36.4	70.8	72.7	67.8
(8, 2, 5, 30)	58.4	84.4	81.9	36.2	72.2	71.0	67.4
(8, 2, 5, 50)	62.5	86.6	87.1	37.6	74.0	76.8	70.8
(8, 2, 8, 10)	63.1	86.7	85.1	38.2	72.9	75.8	70.3
(8, 2, 8, 30)	59.2	84.0	83.9	36.2	72.6	72.8	68.1
(8, 2, 8, 50)	62.3	86.2	87.3	38.6	74.7	77.7	71.1
(8, 5, 2, 10)	62.2	84.3	83.1	37.2	74.7	75.9	69.6
(8, 5, 2, 30)	62.2	85.6	87.3	36.8	74.4	76.2	70.4
(8, 5, 2, 50)	65.9	86.3	87.9	38.2	72.6	78.4	71.5
(8, 5, 8, 10)	65.2	88.0	88.5	38.0	76.5	80.4	72.8
(8, 5, 8, 30)	65.2	87.2	88.7	38.8	74.7	79.0	72.3
(8, 5, 8, 50)	66.0	87.1	88.6	39.2	75.1	81.5	72.9

Table 10: Detailed result of Figure 5b.

Concurrency	$n_e =$ 64	48	32	24	16	12	8	6
2	488	494	510	520	533	544	566	590
3	563	572	635	667	723	767	836	889
4	742	763	839	882	958	1020	1101	1169
6	876	928	1114	1182	1309	1446	1595	1708
8	1060	1144	1367	1455	1614	1779	1947	2072
12	1357	1513	1918	2124	2379	2680	2978	3190
16	1548	1749	2182	2403	2651	2935	3212	3386
24	2022	2338	2922	3265	3621	3870	4203	4351
32	2351	2714	3328	3645	4068	4292	4588	4786
48	2937	3352	3982	4312	4703	4920	5097	5180
64	3679	4203	4931	5321	5766	6011	6155	6280
96	4810	5497	6486	6993	7590	8038	8430	8723
128	5312	5954	6748	7196	7693	7994	8314	8543
192	5729	5836	5959	5909	6027	5999	6122	6171
256	6901	7087	7238	7202	7286	7333	7502	7490
512	8808	9074	9248	9201	9154	9191	9069	9173
784	8853	9147	9280	9481	9571	9607	9285	9356

Table 11: Detailed result of Figure 7.