SMIDT: High-Performance Inference Framework for MoE Models with Dynamic Top-K Routing

Zewen Jin[1 2], Shen Fu[1 2], Chengjie Tang[2 3] Youhui Bai[1*], Shengnan Wang[4], Jiaan Zhu[1 2], Chizheng Fang[1], Ping Gong[1 2], Cheng Li[1 2]

[1]University of Science and Technology of China
[2]Institute of Artificial Intelligence, Hefei Comprehensive National Science Center
[3]Shanxi University
[4]Independent Researcher
{zevin, fushen, andyzhu, fangchizheng, gpzlx1}@mail.ustc.edu.cn, tangcj@sxu.edu.cn,
{youhuibai, chengli7}@ustc.edu.cn, wsn511799@163.com

Abstract

To accelerate Mixture-of-Experts (MoE) inference, the hybrid parallelism paradigm is first applying pipeline parallelism (PP) to vertically divide the model into stages, with each stage further divided horizontally using tensor or expert parallelism. On the algorithm side, dynamic Top-K routing reduces computation by activating fewer experts per token on average. In this paper, we explore the application of dynamic Top-K routing to PP-enabled MoE inference, aiming to fully unleash their combined potential. We identify key performance bottlenecks arising from Top-K value variation across layers, which conflicts with PP's typically uniform stage partitioning, as well as opportunities to optimize memory usage through their integration.

To address these challenges, we present SMIDT, an efficient MoE inference framework tailored for dynamic Top-K routing. SMIDT features: (1) an adaptive, module-level uneven partitioning strategy to balance computation across PP stages, (2) a memory-aware expert replication scheme (DPMoE) that reduces communication overhead, and (3) a lightweight search algorithm combining binary search and dynamic programming to generate efficient parallelism plans. We implement SMIDT on SGLang, a state-of-the-art LLM inference framework, evaluate it on 32 A40 GPUs and 16 A100 GPUs, and compare with manually tuned parallelism strategies. Experimental results show that, when co-locating prefill and decoding phases, SMIDT achieves 1.20–3.13× throughput improvements for prefill-only tasks and 1.05–1.89× for prefill-decoding tasks. When disaggregating prefill and decoding tasks, SMIDT improves average and P99 time-to-first-token (TTFT) by 1.10–1.17× and 1.21–1.26×, respectively.

1 Introduction

Mixture-of-Experts (MoE) has emerged as a standard mechanism to scale large language models (LLMs), due to that it can achieve dramatic parameter growth while only introduces a small amount of computation. Recent frontier models (e.g., DeepSeek-V3 (Liu et al. 2024), Qwen3-235B-A22B (Yang et al. 2025), and Kimi-K2 (MoonshotAI 2025)) exemplified this trend and reported strong results on multilingual and generative benchmarks.

However, efficiently serving MoE models remains both critical and challenging. While several optimizations have been proposed, two main directions have shown promise. First, from the system side, hybrid parallelism is widely adopted. A common approach is to apply pipeline parallelism (PP) to divide the model into stages, each with an even number of layers. Within each stage, tensor parallelism or expert parallelism is used to further improve throughput. Second, from the algorithm side, dynamic Top-K routing has gained popularity for reducing computation during MoE inference while maintaining model quality (Yue et al. 2025; Huang et al. 2024; Zeng et al. 2024). Unlike traditional static Top-K routing (Liu et al. 2024; Yang et al. 2025; MoonshotAI 2025), which assigns a fixed number of experts per token, dynamic routing adjusts the number of active experts based on token characteristics, often resulting in fewer experts being selected on average.

Unfortunately, integrating dynamic Top-K routing into PP-enabled MoE inference introduces significant layer-wise workload variation, leading to compute imbalance and pipeline bubbles that undermine its performance benefits. Our profiling shows that dynamic routing strategies produce highly variable Top-K values across layers, rendering uniform pipeline partitioning ineffective. Meanwhile, time-based partitioning, commonly used to balance computation, results in severe GPU memory imbalance, with some stages often under-utilizing memory resources. For example, more than 30% of GPU memory can be wasted in some stages. While this unused memory could potentially be leveraged to enable layer-wise parallelism that reduces communication or computation cost, such enhancements introduce new compute and memory imbalances, further complicating the framework design.

In this paper, we present SMIDT (**S**trategy for **M**oE **I**nference with **D**ynamic **T**op-K), a high-performance inference framework that realizes an efficient combination of PP-enabled hybrid parallel inference and dynamic Top-K routing. The main contributions are summarized as follows.

- SMIDT adopts the fine-grained partitioning mechanism to balance the time durations across PP stages. In ad-

*Corresponding author.
Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

dition, SMIDT introduces a flexible module-wise parallelism called DPMoE, which trades spare GPU memory for reduced communication gains.

- SMIDT devises an automated, Top-K-aware configuration search algorithm that can navigate a much broader design space, jointly optimizing pipeline partitioning, stage-wise parallelism choices, memory usage, and communication overhead.
- SMIDT is implemented based on SGLang in the experiment. Evaluations on three MoE models are provided to show the superiority of SMIDT, including Qwen2-57B-A14B, Qwen3-235B-A22B and DeepSeek-V2. Specifically, SMIDT achieves up to 3.13× and 1.89× throughput gains for prefill-only tasks and summarization tasks under PD colocation. Under PD disaggregation, SMIDT achieves up to 1.17× improvement in terms of average TTFT for long-decoding tasks.

2 Background

2.1 Mixture-of-Experts and Dynamic Top-K

Mixture-of-Experts. The Mixture-of-Experts (MoE) architecture involves multiple parallel experts composed of feed-forward networks (FFNs) and a learnable router that allocates most related few experts for a given token. It enables dramatic growth in parameter volume while driving only sub-linear compute increases. Recently, DeepSeek-V3 (Liu et al. 2024), Qwen3-235B-A22B (Yang et al. 2025), and Kimi-K2 (MoonshotAI 2025) expanded their model scale using the MoE architecture and achieved significant results on multilingual and generative benchmarks.

Dynamic Top-K techniques. Serving large MoE models is costly. DeepSeek spends about 87k dollars per day for more than 200 H800 nodes to deploy DeepSeek-V3/R1 services (DeepSeek 2024). To mitigate this challenge, recent works have proposed the dynamic Top-K routing algorithms. Compared with the traditional static Top-K routing which allocates the same number of experts for each token, the emerging dynamic Top-K routing adaptively selects suitable numbers of experts for different input tokens. For example, Top-P (Huang et al. 2024) routing adds activated experts for a token until the cumulative gate probability exceeds a threshold. It is demonstrated that such a dynamic routing method can greatly reduce the computation FLOPs by activating less experts on average and provides comparable or higher model quality. For example, Ada-K (Yue et al. 2025) routing adds a lightweight RL-based allocator that decides per-token expert budgets, cutting FLOPs by more than 25% without accuracy loss.

2.2 MoE Inference and Parallelism Strategies

Prefill and decoding. Generally, LLM inference can be divided into two stages, prefill and decoding. In the *prefill* phase, the full prompt is processed in parallel to generate the first token, and the KV states are cached in GPU memory. After this, the *decoding* phase generates subsequent tokens one at a time auto-regressively based on the KV cache and updates it step-by-step.

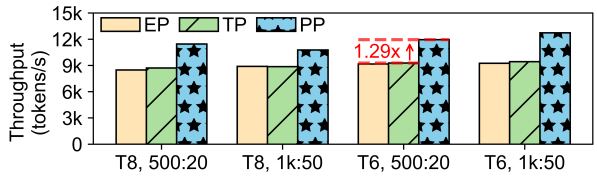

Figure 1: Throughput comparison of different parallelism strategies under fixed Top-K and prefill / decoding lengths. For example, '`T8, 500:20`' means the Top-K value is 8 and the average prompt and output length is 500 and 20.

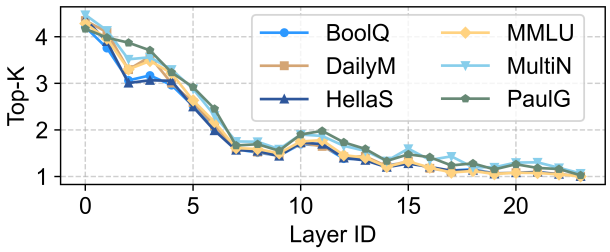

Figure 2: Average Top-K distribution across layers for dynamic Top-K routing.

Parallelism strategies. Several parallelism strategies were introduced for partitioning model parameters and activations. Beyond conventional data parallelism (DP) and tensor parallelism (TP), expert parallelism (EP) was specially designed for MoE models, where each GPU holds a subset of experts, necessitating all-to-all communication for token routing (SGLang Team 2024a; Shoeybi et al. 2020; Jin et al. 2025). Pipeline parallelism (PP) partitions transformer layers into sequential stages, typically with balanced computational overhead (Chen et al. 2025). It processes input mini-batches as overlapped micro-batches, enabling concurrent execution across stages through pipelined scheduling, where each stage operates on different micro-batches simultaneously.

It is widely acknowledged that PP is advantageous in throughput-oriented LLM inference due to its low communication overhead, especially in the prefill phase (Zhong et al. 2024; Agrawal et al. 2025; Zhang et al. 2025). Figure 1 shows the throughput comparison of using PP, TP, and EP to serve Qwen2-57B-A14B with 8 A40 GPUs connected via PCIe. Two summarization tasks are chosen for the workload, one with 1,000-input/50-output tokens on average, and the other with 500-input/20-output tokens. As illustrated in Figure 1, for the fixed Top-K routing, simply adopting naïve equal-division PP can achieve an average speedup of 1.29× and 1.31× against TP and EP, showing the advantage of PP.

3 Motivation

To further optimize inference performance, in this paper, we explore combining dynamic Top-K routing with PP-enabled hybrid parallel execution. Our initial study shows that simply combining the two does not consistently deliver

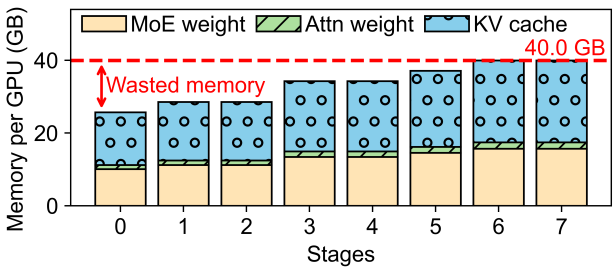

Figure 3: Memory imbalance across stages after balancing based on the stage-wise execution time. Qwen3-235B-A22B is used on 32 A40 GPUs. The threshold of the static memory is set as 40 GB, considering the necessary remaining memory for the intermediate buffers.

the expected performance gains. Instead, we observe noticeable pipeline bubbles during inference, which indicate inefficient utilization of compute resources across stages. The root cause of this inefficiency lies in the compute imbalance across pipeline stages introduced by dynamic top-k routing. Different layers exhibit considerable variation in their top-k selection patterns, leading to heterogeneous computation costs per layer. Figure 2 shows the distribution of the actual Top-K values under Top-P routing (Huang et al. 2024), one of the most common dynamic Top-K routing methods. We can see that the Top-K values vary noticeably across different layers. The similar layer-wise variation pattern also holds for other dynamic Top-K strategies including Ada-K routing (Yue et al. 2025) and AdaMoE (Zeng et al. 2024).

To mitigate the pipeline bubbles and achieve efficient inference performance, we need a suitable parallelism plan with balanced PP partitioning. Considering the significant layer-wise load variation as well as the complicated combination of different parallelism approaches, an automatic search algorithm is further required, which should be aware of the dynamic Top-K routing.

In addition, we observe that simply partitioning the stages based on the time basis incurs severe memory imbalance across pipeline stages. As shown in Figure 3, deploying Qwen3-235B-A22B on 32 GPUs with 8-stage partitioned via a time-based strategy reveals uneven memory allocation, where early stages with less layers have significantly lower memory footprints, leading to the expensive GPU memory under-utilized. For instance, Stage 0 wastes over 37% of its available capacity.

Rather than letting this memory go unused, we consider to innovate layer-wise parallelism for trading memory usage for reduced computation or communication cost, further improving the inference pipeline. However, incorporating such innovations reintroduces compute and memory imbalance across stages, exacerbating the complexity of the parallelism configuration problem. As a result, we need to extend our parallelism configuration search algorithm to jointly consider a much larger search space, consisting of many factors such as PP partitioning choices, dynamic Top-K patterns, layer-wise parallelisms, memory consumption, execution time and communication overhead.

Notation	Description
N	number of layers
S	number of PP stages
L	sequence of attention and MoE modules
$l_{i,a}, l_{i,m}$	attention/MoE module of the i-th layer
$L[b_{j-1} : b_j]$	sequence of attention and MoE modules ranging from b_{j-1} to b_j (not including b_j)
B	sequence of PP segmentation indices
b_i	PP segmentation index, ranging from 0 to $2N$
T, E, D	sequences of TP/EP/DP degree for modules
$t_{i,a}, t_{i,m}$	TP degree for attention/MoE of layer i
$e_{i,a}$	EP degree for attention of layer i (always 1)
$e_{i,m}$	EP degree for MoE of layer i
$d_{i,a}, d_{i,m}$	DP degree for attention/MoE of layer i
$T[b_{j-1} : b_j]$	sequence of TP degrees of modules ranging from b_{j-1} to b_j (not including b_j)
$E[b_{j-1} : b_j]$	sequence of EP degrees of modules ranging from b_{j-1} to b_j (not including b_j)
$D[b_{j-1} : b_j]$	sequence of DP (DPMoE) degrees of modules ranging from b_{j-1} to b_j (not including b_j)
P	total number of GPUs
r	GPUs per stage
M	memory limit per GPU
U	integer set $\{1, \ldots, S\}$
V	integer set $\{1, \ldots, 2N\}$
W	integer set $\{1, \ldots, r\}$

Table 1: Description of the notations used in this paper.

4 SMIDT Design

To address the above challenges, we design SMIDT, an efficient inference framework for MoE models under dynamic Top-K routing, with the following innovations. The detailed notations used in this section are listed in Table 1.

4.1 Fine-Grained Partitioning and Flexible Module-Wise Parallelism

Existing inference frameworks generally partition the model with the entire single transformer layer as the smallest unit when using PP, which is a key factor leading to the inefficiency under dynamic Top-K routing. In SMIDT, we support a finer module-level partitioning so that the inner modules including attention and MoE can be individually assigned to the corresponding stages. This fine-grained partition provides a more flexible operational space for balancing the PP stages, thereby offering more opportunities to enhance the overall efficiency.

Though such a manner enables a time balancing stage partition of PP, as mentioned in Section 3, it further leaves substantial spare memory in some PP stages. Inspired by DPAttention (SGLang Team 2024a), which utilizes spare storage space in exchange for a reduction of communication by replicating attention weights along the DP dimension, we propose a DPMoE strategy which can eliminate communication overhead in a similar manner by replicating MoE weights along the DP dimension. By selectively applying

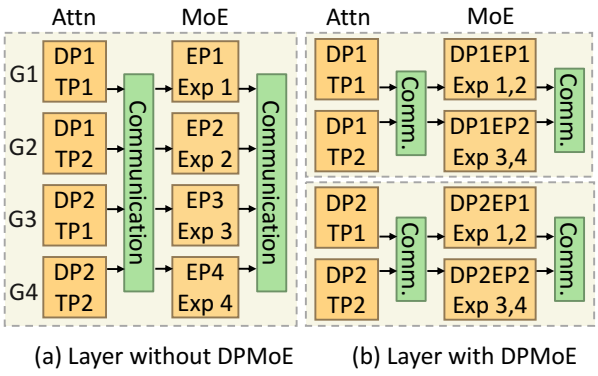

Figure 4: DPMoE illustration with 4 GPUs. For example, 'DP1', 'TP1', 'EP1' refer to the first rank of the DP, TP, and EP parallelism, respectively, while 'Exp 4' refers to the 4th expert.

DPMoE to some MoE layers stored in the GPUs with idle memory, SMIDT achieves significant end-to-end efficiency gains without consuming additional resources. Figure 4 provides a comparison between MoE layers with/without DPMoE for a clearer understanding. As shown in the figure, by applying DPMoE, the heavy communication spanning four GPUs (Figure 4a) is turned into a light communication spanning only two GPUs (Figure 4b).

4.2 Non-Trivial Parallelism Strategy Searching

Next, a key challenge is generating an efficient hybrid parallelism strategy that combines standard parallelisms with the new DPMoE for a given model. Note that both the fine-grained partition and the newly added DPMoE mode significantly expand the search space. The parallelism strategy search becomes a complex task, which is beyond the scope of the existing parallelism search methods (Lin et al. 2025). We need a more effective automatic search algorithm to produce the parallelism strategy in this situation.

In the following, we first formalize the search space and then we define a mixed integer optimization problem for generating a desired parallelism strategy for model partition. After this, we present an efficient search algorithm to solve the problem, followed by a complexity analysis.

Exponential search space. Considering an MoE model with N Transformer layers, each of which has C parallelism strategy choices, there are totally C^N configurations in the search space. In addition, module-wise stage partitioning makes it even more complicated. For a specific PP degree of S, the N layers with $2N$ modules of attention and MoE involve $\binom{2N-1}{S-1}$ choices for fine-grained stage partitioning. The overall count of parallelism strategies for a specific S is $\binom{2N-1}{S-1}C^N$. Efficiently searching for an optimal parallelism strategy with such exponential space is challenging.

Problem definition. Our objective is to determine the optimal module-level partition strategy and module-wise parallelism configuration under diverse PP degrees. In practice, the choices of feasible PP degrees is typically limited, therefore we do not treat the PP degree as a variable to simplify the problem and we can enumerate widely-used PP degrees to search for non-trivial strategies, and then identify the globally desired plan across all candidate PP degrees.

Given an MoE model with N Transformer layers, we define the set $L = [l_{1,a},\ l_{1,m},\ ...,\ l_{N,a},\ l_{N,m}]$ of length $2N$ as its main components, where $l_{i,a}$ and $l_{i,m}$ refer to the attention and MoE modules of the i-th layer. We ignore low-cost modules like normalization. Assume that the degree of PP is S, that is, L is decomposed into S stages $L = [L_1, ..., L_S]$. Define the optimized variables $B = [b_0, b_1, ..., b_S]$ as the PP segmentation points, where b_j are integers from 0 to $2N$ and $0 = b_0 < b_1 < ... < b_{S-1} < b_S = 2N$. Then we can represent L_j by $L_j = L[b_{j-1} : b_j]$ for all j. Each attention and MoE module is further configured with TP, EP, or DP (the EP degree is set as 1 for attention, and DP denotes the DPMoE for MoE). We define the optimized variables $T = [t_{1,a},\ t_{1,m},\ ...,\ t_{N,a},\ t_{N,m}]$, $E = [e_{1,a},\ e_{1,m},\ ...,\ e_{N,a},\ e_{N,m}]$, and $D = [d_{1,a},\ d_{1,m},\ ...,\ d_{N,a},\ d_{N,m}]$ as the degrees of TP, EP, and DP for different modules. Specially, as stated before, $e_{i,a} = 1$ for all i. Then we represent T, E, D by $T = [T_1, ..., T_S], E = [E_1, ..., E_S]$, and $D = [D_1, ..., D_S]$, where $T_j = T[b_{j-1} : b_j], E_j = E[b_{j-1} : b_j]$, and $D_j = D[b_{j-1} : b_j]$ define the specific partition of the modules in stage j.

For stage i, the time duration is related to the specific component L_i and its corresponding inner parallelism strategy, so we represent the time duration of stage i by $Dur(L_i, T_i, E_i, D_i)$. If a specific module is partitioned with degrees t, e, d, then it totally requires $t{\times}e{\times}d$ GPU cards. To simplify the problem, we restrict that each PP stage occupies the same number of GPUs, so $t_{i,a}{\times}e_{i,a}{\times}d_{i,a} = t_{i,m}{\times}e_{i,m}{\times}d_{i,m} = r$ for all i, where $r = P/S$ is the number of GPUs used for each stage and P is the total number of GPUs. For convenience, we use t_i, e_i, d_i to denote the i-th element of T, E, D, respectively in the following. At last, we need to consider the memory. We should ensure that each GPU card will not exceed its memory bound. For each PP stage j with the parallelism strategy T_j, E_j, D_j, we use $Mem(L_j, T_j, E_j, D_j) \in R^r$ to denote the memory consumptions of the corresponding r GPU cards. Based on the above definitions, now we can formulate the optimal model partition problem as follows

$$
\begin{aligned}
&\min_{B,T,E,D} \max_j Dur(L_j, T_j, E_j, D_j), \\
&s.t.\ \max(Mem(L_j, T_j, E_j, D_j)) \le M \quad \forall j \in U \\
&\quad L_j = L[b_{j-1} : b_j], \quad \forall j \in U \\
&\quad T_j = T[b_{j-1} : b_j], \quad \forall j \in U \\
&\quad E_j = E[b_{j-1} : b_j], \quad \forall j \in U \\
&\quad D_j = D[b_{j-1} : b_j] \quad \forall j \in U \\
&\quad 0 = b_0 < b_1 < ... < b_{S-1} < b_S = 2N \\
&\quad t_i{\times}e_i{\times}d_i = r, \quad \forall i \in V \\
&\quad b_j \in V, \quad \forall j \in U, \\
&\quad t_i, e_i, d_i \in W \quad \forall i \in V, \\
&\quad e_i = 1 \quad \text{if} \quad i\%2 = 1
\end{aligned}
\tag{1}
$$

Algorithm 1: **BINARY SEARCH**

Input: N, Transformer layer number; P, Device number; M, Memory limit per device; S, PP degree; eps, Tolerance threshold of duration accuracy, set by default as 0.1 ms.
Output: $Plan$, A concrete parallelism plan for PP degree of S.
$d_{lo} \leftarrow 0$; $d_{hi} \leftarrow Dur(L)$
while $d_{hi} - d_{lo} > eps$ **do**
  $d_{mid} \leftarrow \frac{d_{lo}+d_{hi}}{2}$
  $status, Plan \leftarrow Feasible(d_{mid}, N, P, M, S)$
  if $status$ **is True then**
    $d_{hi} \leftarrow d_{mid}$
  else
    $d_{lo} \leftarrow d_{mid}$
  end if
end while
return $Plan$

Algorithm 2: **FEASIBLE**

Input: D, Time limit; N, Transformer layer number; P, Device number; M, Memory limit per device; S, PP degree.
Output: $Status$, Status of feasibility; $Plan$, A concrete parallelism plan for PP degree of S.
$r \leftarrow P/S$; $left \leftarrow 0$; $right \leftarrow 0$; $stages \leftarrow 0$; $Plan \leftarrow \{\}$
while $left \leq 2N$ **do**
  $duration, StagePlan \leftarrow Knapsack(left, right, r, M)$
  if $right \leq 2N$ **and** $duration \leq D$ **then**
    $right \leftarrow right + 1$
  else
    $\_, StagePlan \leftarrow Knapsack(left, right - 1, r, M)$
    $Plan \leftarrow Plan \bigcup \{StagePlan\}$
    $stages \leftarrow stages + 1$
    $left \leftarrow right$
  end if
end while
if $stages \leq S$ **then**
  return True, $Plan$
end if
return False, **None**

where M is the largest available memory of a GPU card, the integer set $U := \{1, ..., S\}$, $V := \{1, ..., 2N\}$, and $W := \{1, ..., r\}$. Our goal is to minimize the largest time duration in PP stages since it determines the overall efficiency of PP. In addition, we need to ensure that each GPU card does not exceed its memory bound.

To precisely estimate $Dur(\cdot)$ and $Mem(\cdot)$, the solver employs the profiler tools of NVIDIA Nsight Systems (NVIDIA 2025) and PyTorch (PyTorch 2025) to collect the duration and memory footprint information of each module at runtime under different configurations, including the dynamic Top-K distribution and the parallelism combinations of TP, EP, and DP.

Search algorithm. To solve this problem, we design a search algorithm integrating multiple simplifications and optimizations from several observations. First, the optimization problem 1 is a classic min-max problem, allowing us to leverage the binary search method to efficiently find a minimum duration for the slowest PP stage given the PP degree S, as elaborated in Algorithm 1. Here, we adjust the low and high bound of the duration d_{lo} and d_{hi} to make them close to the minimum feasible duration, and d_{hi} is initialized as the duration of all layers. Second, for every duration candidate of binary search, we check its feasibility (Line 6 of Algorithm 1). Note that our goal is to minimize the stage duration under a memory budget, similar to the two-dimensional knapsack problem aiming at higher value under the weight constraint (Mathews 1896). Inspired by the classic dynamic programming algorithm for the knapsack problem (Salkin and De Kluyver 1975), we design the FEASIBLE algorithm to search for the feasible plan in Algorithm 2.

The algorithm searches for the non-trivial module-level partition plan and leverages *Knapsack* function (Line 5) to find optimal module-wise parallelism strategy. Finally, we employ a greedy heuristic to accelerate the dynamic programming searching in Algorithm 2. A native dynamic programming approach calls the *Knapsack* subroutine once for every pair of layer indices (i, j) where $1 \leq i \leq j \leq 2N$, resulting in $O(N^2)$ total calls. We observe a monotonic property that *given a fixed left boundary of a stage, extending the right boundary increases both duration and memory cost*, which inspires us to greedily select the longest feasible interval starting from each position. Based on this insight, we apply a sliding-window scheme to partition the entire sequence of layers into contiguous sub-intervals. This optimization significantly reduces the number of *Knapsack* calls to at most $O(N)$, as only one knapsack evaluation is needed per window. Furthermore, by reusing the results produced through previous *Knapsack*, each layer participates in exactly one stage searching step, resulting in an amortized time complexity of $O(NM)$ for the entire sliding-window partitioning process.

After getting the partition and parallelism strategy for a specific PP degree through Algorithm 1, we decide the optimal PP degree by comparing the estimated throughput of all enumerated candidate PP degrees.

Complexity analysis. The overall time complexity of our search algorithm can be decomposed into two parts. First, the binary search over the limited duration parameter D incurs a factor of $O(\log D)$. Second, for each candidate duration, we invoke the FEASIBLE dynamic programming routine that contributes $O(NM)$ as analyzed above. Therefore, we derive the overall time complexity as $O\big(\log D \times NM\big)$ by multiplying the complexity of the above two components. This complexity is significantly lower than the theoretical upper bound discussed in search space analysis. The algorithm is practically efficient, for example, it only takes less than 0.2 seconds to search for the case of serving the Qwen3-235B-A22B model with 94 layers on 32 GPUs. By contrast, exhaustive search takes 2.9 hours and 23.7 hours for 12 layers and 14 layers on 8 GPUs, respectively.

4.3 System Implementation and Optimizations

We implement SMIDT on SGLang v0.4.9.post1 (SGLang Team 2024b; Zheng et al. 2024), a state-of-the-art LLM inference framework. SMIDT supports two deployment modes: PD disaggregation and colocation, which differ in whether the prefill and decode phases are run on separate

Task	Type	Input:Output Len.
HellaS	Simple QA	200:1
BoolQ		200:1
MMLU		400:1
DailyM	Summarization	1000:50
MultiN		3500:100
PaulG		3500:100
ShareGPT	Conversation	500-2000:100-1000

Table 2: Downstream tasks used in the evaluation. The numbers of prompt length and output length refer to the average length in the workload for benchmarking.

GPUs or the same ones.

Asynchronous PP communication. We find that the SOTA inference frameworks employ synchronous PP communication paradigms, where the current pipeline stage initiates computation for the next batch only after the next stage has successfully received the activations sent from the current stage. This blocking mechanism results in idle waiting and hurts the overall GPU utilization. To address this problem, we implement the asynchronous PP communication in SGLang, enabling communication overhead to be hidden by computation costs, thereby improving inference throughput.

Dynamic Top-K kernel implementation. In SGLang, efficient MoE inference relies on two classes of highly-optimized GPU kernels: high-performance GEMM kernels and permutation kernels (including `moe_align_block_size` and `pre_reorder_triton_kernel`). The permutation kernels perform tensor alignment along the token dimension, serving as a preprocessing step to maximize the efficiency of the subsequent GEMM kernel. However, existing permutation kernels require globally fixed Top-K as input, which makes them incompatible with dynamic Top-K routing, as a result, preventing the utilization of GEMM kernels. To support per-token Top-K while retaining the benefits of the hand-tuned GEMM kernels, we reimplemented the permutation kernels via Triton.

5 Evaluation

5.1 Setup

Platform setup. We adopt SGLang as our baseline framework, which is one of the pioneering MoE inference frameworks (SGLang Team 2024b). The main testbed consists of four servers hosting a total of 32 NVIDIA A40 GPUs (48 GB each), linked internally via PCIe 4.0 × 16 and interconnected across nodes by 100 Gbps InfiniBand. To validate on high-end inter-GPU connections like NVLink, we also evaluate with a two-server cluster equipped with 16 NVIDIA A100 GPUs (80 GB each), which are connected via NVLink 3.0 and 200×8 Gbps InfiniBand.

MoE models and dynamic Top-K. We leverage three large MoE models, Qwen2-57B-A14B (Yang et al. 2024) and Qwen3-235B-A22B (Yang et al. 2025), and DeepSeek-V2 (DeepSeek-AI et al. 2024). We adopt two representative dynamic Top-K routing schemes, Top-P routing and Ada-K routing. For Top-P routing, we directly use its publicly released checkpoints, and for Ada-K routing, since it is not open sourced so far, we adopt our reproduced version.

Datasets. In Table 2, under PD colocation, we adopt six prefill-dominant tasks, where CNN-DailyMail, MultiNews, and PaulGrahamEssays (Hermann et al. 2015; Fabbri et al. 2019; Goel 2024) are summarization tasks, while MMLU, BoolQ, and HellaSwag (Hendrycks et al. 2021; Christopher et al. 2019; Zellers et al. 2019) are question answering tasks. Under PD disaggregation, we evaluate with the ShareGPT dataset (Teams 2023). For each evaluation, we run three passes and report the average number.

5.2 End-to-End Evaluation

Under PD colocation. We first evaluate SMIDT using Qwen2-57B-A14B with a small-scale setup with 8 A40 GPUs. Figure 5 shows the throughput comparison under these tasks using different dynamic Top-K routing techniques. For each task, we compare SMIDT against two manually tuned baselines with and without PP, which are hand-tuned to attain the highest throughput under the corresponding parallelism combination options including DP, PP, TP, and EP, while the optimizations of SMIDT are not considered. Figure 5a refers to the results on prefill-only tasks, where only a single output token is generated based on the prompt. Compared with the baseline without PP, hand-tuned PP achieves a higher throughput, which is consistent with the observation in Section 2. Against two hand-tuned baselines, SMIDT improves the throughput by 1.78-3.08× and 1.17-1.31×, respectively. Although both hand-tuned PP and SMIDT employ the same PP degree of 8, SMIDT brings further improvement through the optimization of parallelism strategy and systematical implementation.

Figure 5b refers to the summarization tasks, which include both prefill and decoding phases. SMIDT search results show a PP degree of 4 is optimal for all workloads except CNN-DailyMail-P, which requires PP 2 and TP 4. Compared with the baselines, SMIDT improves the throughput by 1.71-1.89× and 1.07-1.16×, respectively. We observe that SMIDT's throughput gain is a little lower than in prefill-only scenarios, as PP is not advantageous in the decoding phase (Zhong et al. 2024). However, the two phases need to share the same parallelism strategy under PD colocation. But results show that PP is still a desired choice for the prefill-dominant tasks where the decoding length is short, and, SMIDT can achieve significant efficiency gains.

We further extend to a large-scale setup of 32 A40 GPUs to evaluate the performance of SMIDT by using a much larger model Qwen3-235B-A22B. As shown in Figure 6, we remove the baselines with PP degree less than 4, because such small PP degrees while large TP degree introduce heavy inter-node communication. As the PP degree increases, the throughput of the baselines initially improves but begins to decline when the PP degree exceeds 8, demonstrating that higher PP degrees do not necessarily yield better performance. The optimal PP degree suggested by SMIDT is 8 for all tasks. Compared with all baselines, SMIDT achieves the average speedups 1.28×, 1.46× and 1.62× re-

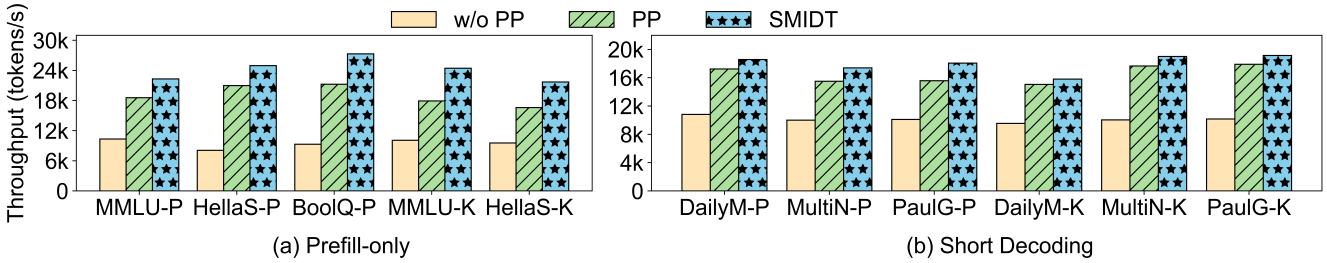

Figure 5: End-to-end throughput comparison of different parallelism strategies under prefill-dominant tasks. The tasks with the '-P' and '-K' suffices refer to Top-P and Ada-K routing, respectively. (a) shows results for prefill-only tasks and (b) shows for short-decoding tasks for summarization. Qwen2-57B-A22B is used on 8 A40 GPUs.

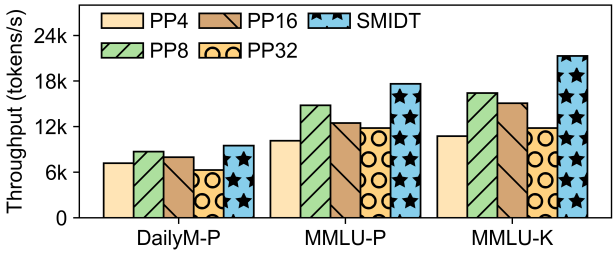

Figure 6: End-to-end throughput comparison of different parallelism strategies. Qwen3-235B-A22B is used on 32 A40 GPUs for large-scale deployment.

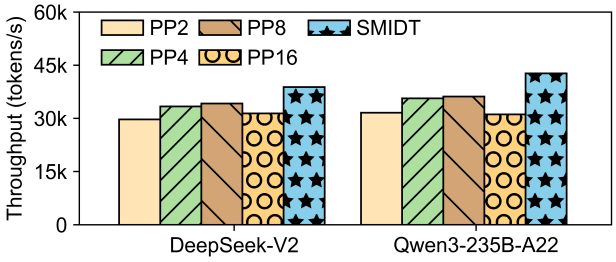

Figure 7: End-to-end throughput comparison on 16 A100 GPUs under the MMLU task. Ada-K routing is used for DeepSeek-V2 and Top-P routing is used for Qwen3-235B-A22B, respectively.

spectively for these three tasks. The speedup stems from SMIDT's capability to adaptively select the best PP degree, and additionally utilize the DPMoE technique to achieve a significant higher throughput under the PD colocation.

Finally, we validate the SMIDT optimization on the high-end NVLink setup with 16 A100 GPUs. Although the intra-server communication bandwidth is much higher with NVLink, it's widely acknowledged that the MoE communication overhead is still a significant bottleneck in distributed serving (Liu et al. 2024). Figure 7 shows the NVLink-based evaluation on DeepSeek-V2 and Qwen3-235B-A22B, where SMIDT still achieves 1.14-1.31× and 1.18-1.37× throughput improvement, respectively.

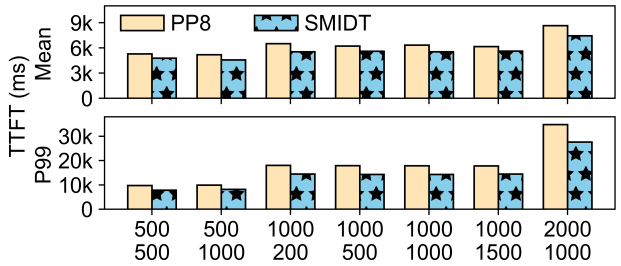

Figure 8: TTFT comparison along different prefill decoding lengths under PD disaggregation. The top and bottom numbers at the x-axis refer to the average prefill and decoding lengths, respectively.

Under PD disaggregation. We further evaluate the latency improvement of SMIDT under PD disaggregation. We allocate 8 GPUs for the prefill and decoding phases, respectively. As suggested by the literates (Zhong et al. 2024; Zhang et al. 2025), we apply PP to the prefill phase and compare the metrics of TTFT (Time-to-First-Token) to focus on the SMIDT optimization on PP. Figure 8 shows the TTFT comparison along different lengths with the ShareGPT dataset. The baseline is configured with PP degree of 8, which achieves the lowest TTFT among the parallelism strategies from manual search. Similar to the findings in PD colocation studies, SMIDT's parallelism strategies consistently outperforms the baseline with different prefill and decoding lengths, achieving mean and P99 TTFT improvements of 1.10-1.17× and 1.21-1.26×, respectively.

5.3 Ablation Study

We break down the contribution of each optimization proposed in Section 4. Figure 9 shows the speedup effect of each SMIDT technique under PD colocation. The baselines correspond to the configurations with the highest throughput when SMIDT optimizations are not applied and are chosen from our manual search. As indicated in Figure 9, SMIDT searching with fine-grained partitioning contributes to the speedup of 10% and 6% for PaulGrahamEssays and MMLU, respectively. DPMoE then contributes to 4% and 14% more speedups for the two tasks. Finally, the asynchronous PP

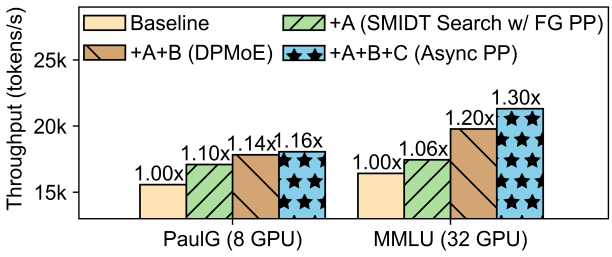

Figure 9: Speedup of each SMIDT optimization technique `A`, `B`, and `C` with the explanation in the parenthesis. `FG PP` refers to fine-grained PP partitioning and `Async PP` refers to asynchronous PP communication. Top-P routing is used.

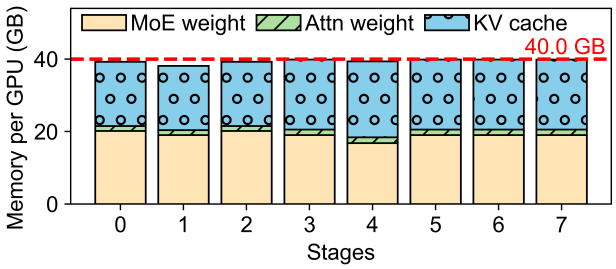

Figure 10: Memory usage per GPU under the SMIDT parallelism strategy with Qwen3-235B-A22B on 32 A40 GPUs.

communication contributes to 2%, and 10% more speedups, respectively. As the number of pipeline stages increases, the blocking effect accumulates across the pipeline and becomes more significant at a higher PP degree. Therefore, asynchronous PP communication provides more contribution to the MMLU workload, for which the optimal PP degree is 8.

5.4 Analysis of Memory Usage

Figure 10 shows the memory usage of each GPU card for the Qwen3-235B-A22B model on 32 GPUs. For a fair comparison with the evaluation in Figure 3, we set the same KV cache budget to hold the same number of tokens in the memory. Compared with the memory usage in Figure 3, the memory imbalance issue is basically addressed after SMIDT optimization and rare memory is wasted.

To understand how the wasted memory is exploited in SMIDT, we further show the specific configurations of the time-based partitioning baseline and SMIDT in Table 3, including the detailed model partition and the average degree of DPMoE at each PP stage. As elaborated in the table, the partition of SMIDT is much more balanced, in terms of the number of attention and MoE modules in each stage. According to the search results given by SMIDT, the degrees of DPMoE of different MoE layers are not necessarily equal. For example, among the twelve MoE layers in Stage 4, the degrees of DPMoE are 1 for nine of them, and 2 for the other three MoE layers. Hence the average DPMoE degree is 1.25 for this stage.

PP Stage	#Attn Modules		#MoE Modules	
	Baseline	SMIDT	Baseline	SMIDT
0	9	11	9 (1.00)	11 (1.64)
1	10	11	10 (1.00)	10 (1.70)
2	10	11	10 (1.00)	12 (1.50)
3	12	12	12 (1.00)	12 (1.42)
4	12	13	12 (1.00)	12 (1.25)
5	13	12	13 (1.00)	12 (1.42)
6	14	12	14 (1.00)	13 (1.31)
7	14	12	14 (1.00)	12 (1.42)

Table 3: The statistics of the attention and MoE modules for each PP stage. The numbers in the parenthesis refer to the average degree of DPMoE across layers in the corresponding PP stage.

6 Related Work

Parallelism plan searching. For training, nnScaler (Lin et al. 2024) constructs a rich, constraint-guided search space through three primitives for generating efficient parallelism plans. For inference, DistServe (Zhong et al. 2024) disaggregates prefill and decoding phases and proposes a simulator to evaluate per-phase combinations of TP and PP parallelism to produce high-throughput plans. For the scenario of PD colocation, Seesaw (Su et al. 2025) proposes a method to dynamically switch the parallelism strategy between prefill and decoding. However, they do not consider the pattern of large MoE models. Apex (Lin et al. 2025) is a simulator to search for the optimal parallelism strategy and helps to decide whether to use EP or TP in MoE models, while it does not support PP for MoE models. Furthermore, all these works do not consider the layer-wise imbalance patterns, rendering them unsuitable for the scenarios targeted by SMIDT. To the best of our knowledge, SMIDT is the first work that systematically addresses the inefficiency in combining PP with dynamic Top-K routing for MoE models.

PP optimizations in LLM inference. Some approaches also address pipeline bubbles caused by stage-level workload imbalance. For example, TD-Pipe (Zhang et al. 2025) temporally separates prefill and decoding phases to eliminate phase-switch bubbles, while gLLM (Guo et al. 2025) uses adaptive token throttling for fine-grained batch scheduling. In contrast, SMIDT tackles the imbalance introduced by combining pipeline parallelism with dynamic Top-K routing, distinct from prior work focused on phase divergence or batch size variation.

7 Conclusion

Powered by an automatic search algorithm, SMIDT resolves key challenges in combining dynamic Top-K routing with pipeline parallelism. It introduces a module-level partitioning scheme and a flexible DPMoE strategy to balance stage durations and fully utilize spare GPU memory. Evaluated on 7 datasets across 3 MoE models using 32 A40 GPUs and 16 A100 GPUs, SMIDT achieves significant improvements in throughput and latency.

Acknowledgements

We thank the anonymous reviewers for their insightful comments. This work is supported by the National Key R&D Program of China under Grant No. 2024YFB4505701. We thank the computing resources provided by Institute of Artificial Intelligence, Hefei Comprehensive National Science Center. We also thank Yinhe Chen and Dongqi Tian for their valuable assistance with the support of PD disaggregation on SGLang.

References

Agrawal, A.; Qiu, H.; Chen, J.; Íñigo Goiri; Zhang, C.; Shahid, R.; Ramjee, R.; Tumanov, A.; and Choukse, E. 2025. Medha: Efficiently Serving Multi-Million Context Length LLM Inference Requests Without Approximations. arXiv:2409.17264.

Chen, R.; Lu, G.; Wang, Y.; Zhang, R.; Hu, Z.; Miao, Y.; Cai, Z.; Leng, J.; and Guo, M. 2025. BAFT: Bubble-aware Fault-tolerant Framework for Distributed DNN Training with Hybrid Parallelism. *Frontiers of Computer Science*, 19(1): 191102.

Christopher, C.; Kenton, L.; Ming-Wei, C.; Tom, K.; Collins, M.; and Kristina, T. 2019. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *NAACL*.

DeepSeek. 2024. Day 6: One More Thing, DeepSeek-V3/R1 Inference System Overview. https://github.com/deepseek-ai/open-infra-index/blob/main/202502OpenSourceWeek/day_6_one_more_thing_deepseekV3R1_inference_system_overview.md#computation-communication-overlapping. [last access: August 1, 2025].

DeepSeek-AI; Liu, A.; Feng, B.; Wang, B.; Wang, B.; Liu, B.; Zhao, C.; Dengr, C.; Ruan, C.; Dai, D.; et al. 2024. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. arXiv:2405.04434.

Fabbri, A. R.; Li, I.; She, T.; Li, S.; and Radev, D. R. 2019. Multi-News: a Large-Scale Multi-Document Summarization Dataset and Abstractive Hierarchical Model. arXiv:1906.01749.

Goel, S. 2024. PaulGrahamEssays (Revision 0c7155a).

Guo, T.; Zhang, X.; Du, J.; Chen, Z.; Xiao, N.; and Lu, Y. 2025. gLLM: Global Balanced Pipeline Parallelism System for Distributed LLM Serving with Token Throttling. arXiv:2504.14775.

Hendrycks, D.; Burns, C.; Basart, S.; Zou, A.; Mazeika, M.; Song, D.; and Steinhardt, J. 2021. Measuring Massive Multitask Language Understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Hermann, K. M.; Kociský, T.; Grefenstette, E.; Espeholt, L.; Kay, W.; Suleyman, M.; and Blunsom, P. 2015. Teaching Machines to Read and Comprehend. In *NIPS*, 1693–1701.

Huang, Q.; An, Z.; Zhuang, N.; Tao, M.; Zhang, C.; Jin, Y.; Xu, K.; Xu, K.; Chen, L.; Huang, S.; and Feng, Y. 2024. Harder Task Needs More Experts: Dynamic Routing in MoE Models. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 12883–12895. Bangkok, Thailand: Association for Computational Linguistics.

Jin, Z.; Wang, S.; Zhu, J.; Zhan, H.; Bai, Y.; Zhang, L.; Ming, Z.; and Li, C. 2025. BigMac: A Communication-Efficient Mixture-of-Experts Model Structure for Fast Training and Inference. arXiv:2502.16927.

Lin, Y.-C.; Kwon, W.; Pineda, R.; and Paravecino, F. N. 2025. APEX: An Extensible and Dynamism-Aware Simulator for Automated Parallel Execution in LLM Serving. arXiv:2411.17651.

Lin, Z.; Miao, Y.; Zhang, Q.; Yang, F.; Zhu, Y.; Li, C.; Maleki, S.; Cao, X.; Shang, N.; Yang, Y.; Xu, W.; Yang, M.; Zhang, L.; and Zhou, L. 2024. nnScaler: constraint-guided parallelization plan generation for deep learning training. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, OSDI'24. USA: USENIX Association. ISBN 978-1-939133-40-3.

Liu, A.; Feng, B.; Xue, B.; Wang, B.; Wu, B.; Lu, C.; Zhao, C.; Deng, C.; Zhang, C.; Ruan, C.; et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Mathews, G. B. 1896. On the Partition of Numbers. *Proceedings of The London Mathematical Society*, 486–490.

MoonshotAI. 2025. Kimi-K2. https://github.com/MoonshotAI/Kimi-K2?tab=readme-ov-file#4-deployment. [last access: August 1, 2025].

NVIDIA. 2025. NVIDIA Nsight Systems. https://docs.nvidia.com/nsight-systems/UserGuide/index.html. [last access: August 1, 2025].

PyTorch. 2025. PyTorch. https://pytorch.org. [last access: August 1, 2025].

Salkin, H. M.; and De Kluyver, C. A. 1975. The knapsack problem: A survey. *Naval Research Logistics Quarterly*, 22(1): 127–144.

SGLang Team, T. 2024a. Data Parallelism Attention For DeepSeek Models. https://lmsys.org/blog/2024-12-04-sglang-v0-4/#data-parallelism-attention-for-deepseek-models. [last access: August 1, 2025].

SGLang Team, T. 2024b. SGLang. https://github.com/sgl-project/sglang. [last access: August 1, 2025].

Shoeybi, M.; Patwary, M.; Puri, R.; LeGresley, P.; Casper, J.; and Catanzaro, B. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053.

Su, Q.; Zhao, W.; Li, X.; Andoorveedu, M.; Jiang, C.; Zhu, Z.; Song, K.; Giannoula, C.; and Pekhimenko, G. 2025. Seesaw: High-throughput LLM Inference via Model Resharding. arXiv:2503.06433.

Teams, S. 2023. ShareGPT. [last access: August 1, 2025].

Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yang, A.; Yang, B.; Hui, B.; Zheng, B.; Yu, B.; Zhou, C.; Li, C.; Li, C.; Liu, D.; Huang, F.; et al. 2024. Qwen2 technical report. *arXiv:2407.10671*.

Yue, T.; Guo, L.; Cheng, J.; Gao, X.; Huang, H.; and Liu, J. 2025. Ada-K Routing: Boosting the Efficiency of MoE-based LLMs. In *The Thirteenth International Conference on Learning Representations*.

Zellers, R.; Holtzman, A.; Bisk, Y.; Farhadi, A.; and Choi, Y. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Zeng, Z.; Miao, Y.; Gao, H.; Zhang, H.; and Deng, Z. 2024. AdaMoE: Token-Adaptive Routing with Null Experts for Mixture-of-Experts Language Models. arXiv:2406.13233.

Zhang, H.; Wei, T.; Zheng, Z.; Du, J.; Chen, Z.; and Lu, Y. 2025. TD-Pipe: Temporally-Disaggregated Pipeline Parallelism Architecture for High-Throughput LLM Inference. arXiv:2506.10470.

Zheng, L.; Yin, L.; Xie, Z.; Sun, C.; Huang, J.; Yu, C. H.; Cao, S.; Kozyrakis, C.; Stoica, I.; Gonzalez, J. E.; Barrett, C.; and Sheng, Y. 2024. SGLang: Efficient Execution of Structured Language Model Programs. arXiv:2312.07104.

Zhong, Y.; Liu, S.; Chen, J.; Hu, J.; Zhu, Y.; Liu, X.; Jin, X.; and Zhang, H. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. arXiv:2401.09670.