AdaFuse: Accelerating Dynamic Adapter Inference via Token-Level Pre-Gating and Fused Kernel Optimization

Qiyang Li[1], Rui Kong[1], Yuchen Li[1]*, Hengyi Cai[1], Shuaiqiang Wang[1], Linghe Kong[2], Guihai Chen[2], Dawei Yin[1]

[1]Baidu Inc.
[2]Shanghai Jiao Tong University

Abstract

The integration of dynamic, sparse structures like Mixture-of-Experts (MoE) with parameter-efficient adapters (e.g., LoRA) is a powerful technique for enhancing Large Language Models (LLMs). However, this architectural enhancement comes at a steep cost: despite minimal increases in computational load, the inference latency often skyrockets, leading to decoding speeds slowing by over 2.5 times. Through a fine-grained performance analysis, we pinpoint the primary bottleneck not in the computation itself, but in the severe overhead from fragmented, sequential CUDA kernel launches required for conventional dynamic routing. To address this challenge, we introduce AdaFuse, a framework built on a tight co-design between the algorithm and the underlying hardware system to enable efficient dynamic adapter execution. Departing from conventional layer-wise or block-wise routing, AdaFuse employs a token-level pre-gating strategy, which makes a single, global routing decision for all adapter layers before a token is processed. This "decide-once, apply-everywhere" approach effectively staticizes the execution path for each token, creating an opportunity for holistic optimization. We capitalize on this by developing a custom CUDA kernel that performs a fused switching operation, merging the parameters of all selected LoRA adapters into the backbone model in a single, efficient pass. Experimental results on popular open-source LLMs show that AdaFuse achieves accuracy on par with state-of-the-art dynamic adapters while drastically cutting decoding latency by a factor of over 2.4x, thereby bridging the gap between model capability and inference efficiency.

Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in language understanding and generation, enabling significant progress in a wide range of tasks, including conversational AI (Li et al. 2025a; Chen et al. 2025; Li et al. 2023d, 2025d; Wei et al. 2025), code generation (Xiong et al. 2024; Wang et al. 2024), search (Li et al. 2025f; Liao et al. 2023; Li et al. 2025e,b,c), and recommendation (Li et al. 2023c,b; Lu and Yin 2025; Cui et al. 2025b,a; Liu and Lu 2025; Mo et al. 2025). To customize the pretrained models to vertical domains or further enhance their capabilities, various adapter techniques such as Low-Rank Adapters (LoRA) (Hu et al. 2021), LLaMA-Adapter (Zhang et al. 2023), and Prompt Tuning (Lester, Al-Rfou, and Constant 2021a; Li et al. 2023a; Tong et al. 2025; Liao, Chu, and Wang 2024) have been employed with great success. These methods are known for improving the performance of LLMs on downstream tasks without requiring extensive retraining, thereby enabling efficient model adaptation and customization.

Among these approaches, dynamic adapters (Feng et al. 2024; Gou et al. 2024; Luo et al. 2024) represent an even more potent strategy to augment the capacity of adapters. Unlike static adapters that apply the same transformation to all inputs, dynamic adapters conditionally activate different sets of lightweight modules based on input characteristics, enabling more sophisticated and context-aware model behavior. By integrating conditionally computed lightweight adapters into the pretrained model, dynamic adapters allow for selective fine-tuning of adapter parameters. This technique not only maintains the original strengths of the model but also substantially increases its adaptability and capacity across diverse tasks and domains. The flexibility to adapt different model components for different inputs theoretically provides superior performance compared to static approaches, making dynamic adapters particularly attractive for multi-task and multi-domain scenarios.

However, we found that despite the relatively minor impact of dynamic adapters on parameter size and computing complexity (typically adding only 1-5% of the origin model), they may introduce significant latency overhead. For instance, the dynamic adapters that we studied all increase decoding inference latency by 250-950%. The seemingly modest computational complexity of the low-rank matrices employed results in substantial extra CUDA kernel execution latency, surpassing that of models without dynamic adapters. This dramatic increase in latency is primarily attributed to the prolonged execution time of context operations during CUDA kernel runs, which considerably exceeds the actual computation time. The fundamental issue lies in the mismatch between the algorithmic design of dynamic adapters and the underlying GPU architecture, where frequent kernel launches and memory access patterns create substantial overhead that outweighs the computational benefits. Dynamic adapters often require four or more ad-

*Corresponding Author

ditional CUDA kernel calls for each layer, in stark contrast to just a single call needed for the forward computation of the original backbone matrix. This excessive number of context operations substantially amplifies the latency overhead, leading to a severe escalation of inference latency.

Reducing the inference latency overhead of dynamic adapters is challenging. Existing dynamic adapters (Dou et al. 2024; Feng et al. 2024; Gao et al. 2024; Gou et al. 2024; Li et al. 2024; Liu et al. 2023; Luo et al. 2024; Wu, Zheng, and Yu 2024a; Yang et al. 2024) adopt block-wise or layer-wise routing structures. This architectural choice inherently requires routing decisions to be made sequentially at each block or layer, preventing the efficient pre-merging of adapters into the backbone weights—a technique successfully used by static LoRA (Hu et al. 2021). The need to dynamically select and compute adapters at different stages of the model introduces a fundamental trade-off: the increased model expressiveness comes at the cost of fragmented, high-latency computations that are difficult to optimize with current system approaches. This makes it prohibitively costly to reduce the inference latency without altering the core design of dynamic routing.

Our Approach: A System-Algorithm Co-Design. Our approach to addressing the challenge is based on a holistic system-algorithm co-design. Specifically, we have developed a MoE-based dynamic adapters structure that facilitates token-wise adapter routing. Each token is associated with k weighted paths of LoRA adapters, activated prior to the decoding of the token. This setup ensures that, although the model is enhanced with dynamic structures, the inference process for each token remains relatively static due to the pre-determined adapters. To further enhance the efficiency, we pre-merge the activated LoRA adapters into the pretrained model's backbone before each token's decoding. This strategy fundamentally reduces the CUDA kernel execution overhead, thereby significantly lowering latency. With this innovative setup, we have re-engineered the inference process to seamlessly switch and merge adapters for each token, aligning the process closely with the original pretrained LLM's token decoding. Another pivotal component of our system is the development of a fused CUDA kernel, named SGMM, which efficiently manages the activated and inactivated adapters. This engineering solution ensures a smooth integration of dynamic adapters, optimizing both performance and efficiency. We evaluate our AdaFuse design across a range of benchmarks, comparing it against multiple state-of-the-art dynamic adapter baselines. The experiment results demonstrate that our approach are comparable with well-established strong baselines. Notably, our method significantly reduces the running overhead associated with other dynamic adapter alternatives, achieving an average speedup of 2.4 times in decoding latency. In summary, our contributions are as follows:

- We uncover the high latency overhead introduced by dynamic adapters, which is a practical issue usually neglected by existing approaches. We analyze the fundamental reasons behind such high overhead, providing insights on the computational bottlenecks.
- We introduce a novel architecture for dynamic adapters, named AdaFuse. This design enhances the capacity of LLM adapters while minimizing the latency overhead, thereby offering an optimal balance between performance and efficiency.
- Through extensive experiments, we demonstrate that AdaFuse not only achieves accuracy on par with existing dynamic adapters across a variety of general and domain-specific tasks, but it also cuts down decoding inference latency by more than 2.4 times.

Background and Motivation

Dynamic Adapters. Given the strengths of both the Mixture of Experts (MoE) (Jiang et al. 2024; Snowflake AI Research Team 2024; The Mosaic Research Team 2024; xAI 2024) and Low-Rank Adaptation (LoRA) (Hu et al. 2021), their integration has become a focal point of recent research efforts. Recent studies (Feng et al. 2024; Gao et al. 2024; Gou et al. 2024; Liu et al. 2023; Luo et al. 2024) have explored combining these two techniques to further augment the capabilities of large language models (LLMs). This integration leverages the scalability of MoE and the efficiency of LoRA, proposing a promising pathway to meet the escalating demands for model performance and efficiency.

Formally, the computation process of dynamic adapters can be formulated as:

$$y^l = f^l(x^l) + \sum_{i=1}^{N} G^l(x^l)_i E_i^l(x^l), \tag{1}$$

where the superscript l means l-th layer, N represents number of adapters experts, $G^l(x^l) = \text{Softmax}(\text{TopK}(W_g^l x^l))$ represents the top-k (typically top-2) router in the dynamic adapters block, f^l represents the pretrained backbone in l-th layer, and $E^l(x^l) = W_{up}^l(W_{down}^l(x^l))$ represents the output of LoRA experts.

Despite an increase in parameters, the experts of the dynamic adapters are activated sparsely, implying that only a limited subset of experts is used per input token. This sparse activation mechanism maintains computational efficiency while significantly expanding the model's capacity to handle diverse scenarios.

Unexpected Latency Overhead of Dynamic Adapters. Although dynamic adapters can enhance accuracy and involve only a modest increase in parameter size and computing complexity, they unfortunately introduce a substantial inference latency overhead. We evaluate different dynamic adapter methods with Llama2-7B (Touvron et al. 2023) on the ShareGPT (OpenChat 2023) dataset for 50 queries one by one, and generate 200 new tokens for each query. As demonstrated in Table 1, existing methods involving dynamic adapters result in an approximate 1%-5% increase in parameter count and less than a 1% increase in computing complexity measured in FLOPS. However, these enhancements lead to a substantial increase in decoding latency, with overheads ranging from 200% to 950%.

To elucidate the sources of latency overhead introduced by dynamic adapters, we conducted a granular analysis of latency within various components during the decoding phase.

Method	Decoding latency (ms/token)	Parameter size (B)	FLOPS (G)
Llama2-7B	2.4	6.74	6.61
MOLA (Gao et al. 2024)	25.3 (+954%)	7.07 (+4.89%)	6.65 (+0.61%)
PESC (Wu, Zheng, and Yu 2024b)	8.5 (+254%)	6.97 (+3.41%)	6.64 (+0.45%)
MoRAL (Yang et al. 2024)	8.6 (+258%)	6.97 (+3.41%)	6.67 (+0.91%)

Table 1: Inference cost of different dynamic adapters.

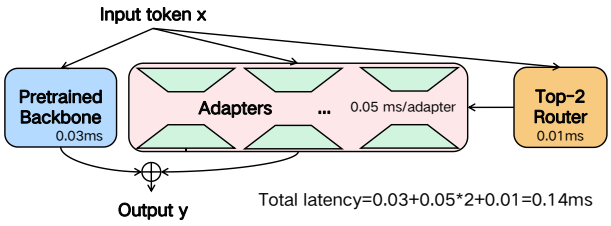


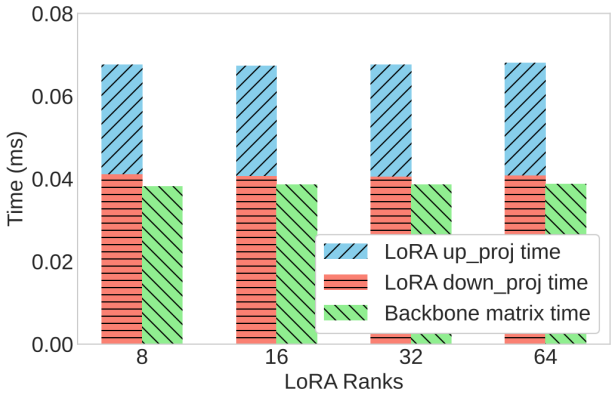
Figure 1: Decoding phase execution time profiling of one dynamic adapter layer in MoRAL. The execution time results were preceded by a warm-up of 100 executions and are obtained on the average of 300 executions.

As illustrated in Figure 1, it is evident that the execution time for the adapters (0.05ms) exceeds that of the pretrained backbone (0.03ms). Despite the relatively modest computational complexity of the LoRA adapters employed in dynamic configurations, each adapter necessitates dual launches of CUDA kernel context operations. The execution time of these CUDA kernels does not correlate linearly with the size of the matrices involved, leading to considerable latency in the adapter components.

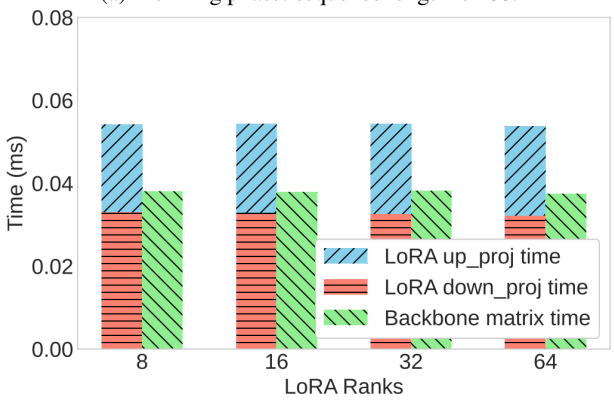
In-depth Latency Profiling. To meticulously investigate how the inference latency of LoRA adapters and the pretrained backbone varies with increasing computational demands, we conducted tests across different input sequence lengths and examined the relationship between adapter rank and inference latency.

As shown in Figure 2, regardless of whether it is during the prefilling or decoding phase, and irrespective of the LoRA ranks being high or low, the latency of the LoRA adapters consistently exceeds that of the backbone. This phenomenon is primarily attributed to the number of CUDA kernel calls rather than the computing complexity involved in each call. The underlying reason is that the latency associated with CUDA kernel calls does not scale linearly with computing complexity. This insight highlights a crucial aspect of system behavior that significantly impacts the performance of dynamic adapters.

Challenge of Reducing Latency Overhead. A straightforward way to reduce inference latency overhead is to reduce the times of CUDA kernel context operations. Like LoRA (Hu et al. 2021), one could pre-merge adapters into the original matrix and then perform token decoding computation. We use this simple strategy in MoRAL by directly merging activated adapters layer by layer before computing. However, the additional operations introduced higher latency, where the decoding latency is 4.5 ms/token, which



(a) Prefilling phase: sequence length is 100.



(b) Decoding phase: sequence length is 1.

Figure 2: Latency breakdown of one dynamic adapter layer under different settings.

is still 88% higher than the original LLM model. This is because merging a LoRA adapter into the backbone matrix requires an additional invocation of a CUDA kernel to perform the matrix multiplication for the up and down projections.

Related Work

Parameter Efficient Fine-tuning (PEFT). Parameter-efficient fine-tuning has emerged as the predominant approach for adapting pretrained large language models (LLMs) to downstream tasks. Representative methods include Adapters (Houlsby et al. 2019), Prefix Tuning (Li and Liang 2021), Prompt Tuning (Lester, Al-Rfou, and Constant 2021b), and LoRA (Hu et al. 2021). Among these, LoRA has gained prominence due to its elegant low-rank decomposition strategy that enables efficient inference through adapter

merging. However, traditional LoRA is limited to static single-adapter scenarios, lacking the dynamicity needed for multi-task adaptation.

Dynamic Adapters and MoE-based PEFT. The convergence of PEFT and MoE principles has led to dynamic adapter architectures. Block-wise methods such as MOLA (Gao et al. 2024) and MoELoRA (Luo et al. 2024) require runtime routing within each block, preventing efficient batching. Layer-wise approaches like MoRAL (Yang et al. 2024) and LoRAMoE (Dou et al. 2024) apply routing at layer granularity but fundamentally cannot achieve the inference efficiency of static adapters due to their layer-by-layer routing decisions. In contrast, our token-wise pre-gated approach makes all routing decisions upfront, enabling adapter merging before inference and eliminating computational overhead entirely.

System Optimizations for LLM Inference. System-level optimizations for LLMs include memory management techniques like PagedAttention (Kwon et al. 2023), quantization (Frantar et al. 2022), and pruning (Frantar and Alistarh 2023). PEFT-specific optimization (Ye et al. 2023) targets static adapter configurations but cannot address dynamic adapter selection due to irregular computation patterns. Our work uniquely addresses the gap between algorithmic innovations in dynamic adapters and system-level optimizations by redesigning the dynamic adapter paradigm to be system-friendly while preserving expressiveness.

Design of AdaFuse

Overview

As shown in Figure 3, given a pre-trained LLM, we first extend it with AdaFuse to enhance its model capacity and then finetune it on either general or domain-specific datasets. The core innovation of AdaFuse lies in its token-wise pre-gated architecture, which fundamentally differs from existing layer-wise or block-wise dynamic adapter approaches. Instead of making routing decisions at each layer independently, AdaFuse employs a single router at the first layer to determine expert activation patterns for all subsequent layers, thereby eliminating the computational overhead of repeated routing computations.

During the decoding phase, each token is initially processed through a router to compute the gating for each layer. The key insight is that once we determine which experts should be activated for a given token, this decision can be propagated across all layers without requiring additional routing computations. This design choice is motivated by our observation that tokens with similar semantic properties tend to activate consistent expert patterns across different layers. To implement this efficiently, we developed an SGMM kernel that facilitates fast fused adapter switching. This advanced functionality enables the rapid merging of activated adapters into the backbone and the efficient unmerging of inactivated adapters from the backbone, significantly reducing the number of CUDA kernel calls.

Model Structure

In AdaFuse, we extend adapters only in the linear layers of the pre-trained backbone, and we insert a Top-2 router G^1 only at the first expanded linear layer. This architectural choice is carefully designed to balance computational efficiency and model expressiveness.

Router Architecture. The Top-2 router G^1 consists of a lightweight linear transformation followed by a softmax activation and top-k selection mechanism. The router takes the hidden representation from the first layer as input and produces routing weights that determine expert activation across all layers.

Expert Configuration. Each expert in AdaFuse is implemented as a LoRA module with rank r typically set to 64 or 128, depending on the model size and task complexity. The number of experts N is configurable and typically ranges from 4 to 16, providing a good balance between model capacity and computational overhead.

Finetuning Phase. As shown in Figure 3 (a), during finetuning, the computation process can be written as:

$$y^l = f^l(x^l) + \sum_{i=1}^{N} G^1(x^1)_i E_i^l(x^l) \tag{2}$$

where AdaFuse only replaces the $G^l(x^l)$ with $G^1(x^1)$ in Equation 1, where x^1 denotes the input in the first expanded linear layer. AdaFuse employs a token-wise pre-gated LoRA structure, meaning that the routing weights for all layer adapters are identical. This design not only preserves the model's dynamicity but also facilitates latency optimization during inference.

Decoding Phase. As depicted in Figure 3 (b), AdaFuse leverages the token-wise pre-gated structure to reduce latency during the decoding phase. During the decoding phase of LLM generation, where the input x is a single token, the Top-2 router G^1 determines which experts' adapters are activated. Then the activated experts are merged into pre-trained backbone. Finally, the fused backbone performs forward process and the decoding computation as:

$$y^l = f_*^l(x^l) \tag{3}$$

To efficiently calculate f_*^l for all layers, we propose to perform fused adapter switching with the SGMM kernel, which merges the parameters of all activated expert adapters across all layers into the original parameters of the pre-trained model in a single CUDA kernel operation. Finally, the fused backbone execute just like the initial pretrained backbone as shown in Equation 3.

Prefilling Phase. The latency of AdaFuse during the prefilling phase is comparable to that of existing dynamic adapters. For the prefilling phase, we have not implemented specific optimizations, as the latency in the LLM generation stage primarily originates from the decoding phase.

Fused Adapter Switching

To calculate $f_*^l = f^l + \sum_{i=1}^{N} G^1(x^1)_i E_i^l$ according to Equation 1 and Equation 2, we may merge multiple experts adapters into backbone because one input token may

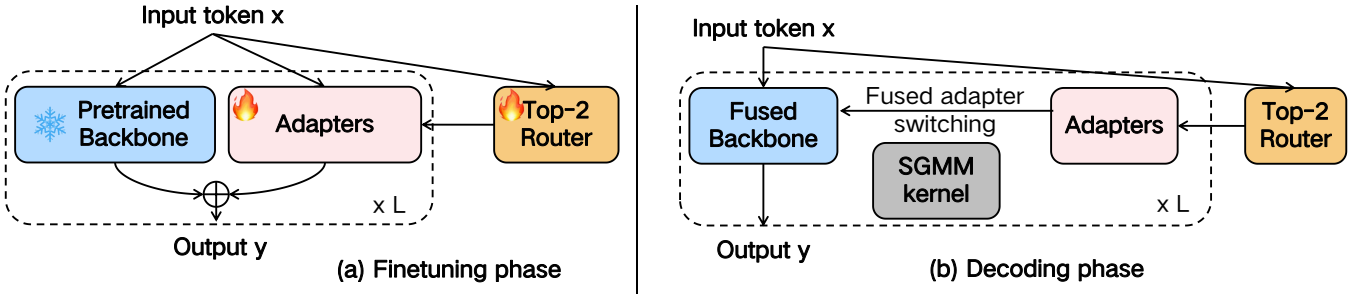


Figure 3: Overview of AdaFuse.

activate multiple adapters (typically Top-2). The experts in AdaFuse are LoRA adapters, which contain down projection LoRA_DOWN and up projection LoRA_UP.

We only need to invoke the CUDA kernel k times instead of N times, as $G^1(x^1)$ specifically targets the top-k selections. To further reduce the number of CUDA kernel calls, we concatenate all LoRA adapters before merging them into the original model parameters as:

$$\text{LoRA\_DOWN}^l = \text{concat}\big[G^1(x^1)_i \cdot \text{LoRA\_DOWN}_i^l, i = 1, \ldots, N\big] \quad (4)$$

$$\text{LoRA\_UP}^l = \text{concat}\big[\text{LoRA\_UP}_i^l, i = 1, \ldots, N\big] \quad (5)$$

Thus, we can merge the concatenated LoRA adapters into origin backbone as:

$$f_*^l = f^l + \text{LoRA\_DOWN}^l \times \text{LoRA\_UP}^l \quad (6)$$

We observe that during the decoding phase, the activated adapters varying from one token to the next. A simple way to obtain f^l is to unmerge the activated adapters by the last token in the current iteration. Then we concatenate the activated adapters of current input and inactivated adapters of last input as:

$$\text{Fused\_LoRA\_DOWN}^l = \text{concat}\big[-(\text{LoRA\_DOWN}^l)^{t-1}, (\text{LoRA\_DOWN}^l)^t\big] \quad (7)$$

$$\text{Fused\_LoRA\_DOWN}^l = \text{concat}\big[-(\text{LoRA\_DOWN}^l)^{t-1}, (\text{LoRA\_DOWN}^l)^t\big] \quad (8)$$

So the concatenated LoRA adapter switching operation can be rewritten as:

$$(f_*^l)^t = (f_*^l)^{t-1} + \text{Fused\_LoRA\_DOWN}^l \times \text{Fused\_LoRA\_UP}^l \quad (9)$$

To calculate Equation 9, we utilize our efficiently designed CUDA kernel, SGMM, to seamlessly integrate these fused adapters into the pretrained LLM backbone for all layers with only one CUDA kernel call.

SGMM Kernel

The straight-forward way to merge the LoRA adapter into the backbone is to merge them layer by layer. This approach requires multiple calls to the CUDA kernel, which introduces additional latency due to kernel launches. Moreover, smaller kernel computations underutilize GPU thread blocks, leading to a low GPU throughput. We observe the layer-by-layer merging operations can be handled concurrently and introduce a CUDA kernel called Segmented Gather Matrix Multiplication (SGMM) to finally handle the merging of LoRA adapters of AdaFuse, adapted from the concept of SGMV proposed by Punica (Chen et al. 2023).

SGMM is designed to execute a batched GEMM operations, which can be summarized by the following equation:

$$f_* = f + \text{Fused\_LoRA\_DOWN} \times \text{Fused\_LoRA\_UP}, \quad (10)$$

where f_* is the resultant updated matrix of the backbone; f is the original weight matrix of the backbone; Fused_LoRA_DOWN and Fused_LoRA_UP are the adapter matrices for weight matrix f. The addition operation within the SGMM kernel is performed in place, significantly reducing the additional memory overhead.

When wrapping these operations into a single CUDA kernel, taking full advantage of the GPU's computational resources is challenging. To achieve this, we divide the matrix multiplication into multiple GEMM tiles and assign them to different thread blocks. On the other hand, these thread blocks must switch context with global memory/shared memory frequently, thus causing significant latency. To tackle this, we adopt a pre-fetch buffer mechanism to hide loading latency.

The input parameters for SGMM are arrays of pointers to the LoRA matrices and the backbone matrices, which respectively store the corresponding entries of each layer's LoRA matrix and the shape of each matrix segment. When launching the kernel, SGMM applies as many thread blocks as possible and divides the large matrix multiplication into multiple GEMM tiles of the same shape, with each tile operating matrix computation. The optimal tiling scheme related to hardware is selected to ensure the full utilization of each thread block. This parallel execution enables the efficient merging and unmerging of LoRA weights, a critical operation in our approach.

Evaluation

Experiment Setup

Datasets/benchmarks. To demonstrate the proposed AdaFuse could augment general ability of LLM, we follow

PESC (Wu, Zheng, and Yu 2024b) and simultaneously fine-tuned the model on a diverse set of skills, including encompassing coding, mathematical, and other general abilities from various subjects. This training involved integrating three distinct datasets from varied domains during the instruction tuning phase: SlimORCA (Lian et al. 2023), Magicoder (Wei et al. 2023), and MetaMathQA (Yu et al. 2023) datasets. We utilize LM-Eval-Harness (Gao et al. 2023) as tool to evaluate general ability on ARC (Clark et al. 2018), HellaSwag (Zellers et al. 2019), MMLU (Hendrycks et al. 2021), TruthfulQA (Lin, Hilton, and Evans 2022), WinoGrande (Sakaguchi et al. 2019), and MT-Bench (Zheng et al. 2023) benchmarks and report the accuracy. Also, to demonstrate the proposed AdaFuse could improve domain specific ability of LLM, we follow MoLA (Gao et al. 2024) and fine-tuned the model on downstream task. We evaluate three recent question-answering benchmarks, including ScienceQA(Lu et al. 2022), CommonsenseQA(Talmor et al. 2019), and OpenbookQA(Mihaylov et al. 2018). To evaluate runtime efficiency performance of AdaFuse, we utilize real-world sharegpt (OpenChat 2023) dataset to simulate user queries. We serve 50 queries from sharegpt dataset one by one, and generate 200 new tokens for each query.

Baselines. We compare AdaFuse with four PEFT approaches, including LoRA (Hu et al. 2021), layer-wise gating dynamic adapters like MoRAL (Yang et al. 2024) and MOLA (Gao et al. 2024), and block-wise gating dynamic adapters PESC (Wu, Zheng, and Yu 2024b). We use LLama2-7B (Touvron, Martin, and et al. 2023) and Mistral-7B (Jiang et al. 2023) as the pretrained base LLM.

Accuracy Evaluation

We evaluate the accuracy of AdaFuse on both general capabilities and domain-specific tasks to demonstrate its effectiveness compared to existing dynamic adapter methods.

General Capability Enhancement. To assess the general capability improvement, we fine-tune AdaFuse and baseline methods on a mixture of general datasets and evaluate on standard benchmarks. Table 2 shows the results across five common evaluation tasks.

Method	ARC	HellaSwag	MMLU	TruthfulQA	Winogrande	Avg
Llama2-7B (base)	51.71	**77.74**	48.30	45.31	72.45	59.10
LoRA	51.79	77.02	50.46	45.13	73.80	59.64
MoRAL (layer-wise)	52.13	77.57	51.10	45.93	**74.35**	60.22
PESC (block-wise)	**53.58**	77.27	51.07	46.04	74.27	**60.45**
AdaFuse (ours)	52.39	77.60	**51.15**	**46.15**	73.32	60.12

Table 2: Accuracy of incremental training achieved with different dynamic adapters.

As shown in Table 2, AdaFuse achieves competitive performance with an average accuracy of 60.12%, which is comparable to the best performing baseline PESC (60.45%). Notably, AdaFuse achieves the highest scores on MMLU and TruthfulQA benchmarks, demonstrating its effectiveness in knowledge-intensive tasks.

Domain-Specific Customization. We further evaluate AdaFuse on domain-specific tasks to assess its adaptation capability. Tables 3 and 4 present results on question-answering tasks for Llama2-7B and Mistral-7B respectively.

Methods	ScienceQA	CommonsenseQA	OpenbookQA	Avg
Llama2-7B (base)	53.19	47.82	45.80	48.94
Full-Parameter	93.12	77.48	80.40	83.67
LoRA	91.01	75.51	77.00	81.17
MoLA (layer-wise)	**91.91**	77.89	**82.80**	**84.20**
MoRAL (layer-wise)	90.74	76.41	76.60	81.25
PESC (block-wise)	90.02	76.00	78.40	81.47
AdaFuse (ours)	91.39	**79.03**	80.40	83.60

Table 3: Accuracy of domain-specific fine-tuning achieved with different dynamic adapters on Llama2-7B.

Methods	ScienceQA	CommonsenseQA	OpenbookQA	Avg
Mistral-7B (base)	62.24	58.93	57.8	59.66
LoRA	94.15	79.85	84.2	86.06
MoRAL (layer-wise)	93.79	81.57	85.8	87.05
PESC (block-wise)	94.33	80.46	86.4	87.06
AdaFuse (ours)	93.82	81.29	86.6	**87.24**

Table 4: Accuracy of domain-specific fine-tuning achieved with different dynamic adapters on Mistral-7B.

For domain-specific tasks, AdaFuse demonstrates strong performance, achieving 83.60% average accuracy on Llama2-7B and 87.24% on Mistral-7B, both competitive with or exceeding baseline methods. Particularly noteworthy is AdaFuse's superior performance on CommonsenseQA, suggesting effective reasoning capability enhancement.

Runtime Performance

Beyond accuracy, a critical evaluation metric for dynamic adapters is their runtime efficiency. We conduct comprehensive latency and memory usage analysis to demonstrate the practical advantages of AdaFuse over existing methods.

As reported in Table 5, we evaluated the inference latency and peak GPU memory usage of our method and various baseline methods on the ShareGPT dataset. The results show that AdaFuse exhibits significantly lower decoding latency compared to all other dynamic adapter methods, being 2.7 times faster than the previously fastest method, PESC. Furthermore, AdaFuse's decoding latency is less than 30% higher than that of the original Llama2-7B model. The dramatic latency reduction achieved by AdaFuse can be attributed to our fundamental architectural innovation. While traditional dynamic adapters require separate routing computations and adapter activations at each layer, our approach consolidates these operations into a single pre-gating decision followed by efficient batch processing through the SGMM kernel. This transforms the inference pattern from multiple sequential operations to a single parallelized computation.

Method	Latency (ms/token)	Memory (GiB)
Llama2-7B	2.4	12.9
MOLA (layer-wise)	25.3 (+954%)	26.3 (+104%)
PESC (block-wise)	8.5 (+254%)	13.1 (+2%)
MoRAL (layer-wise)	8.6 (+258%)	13.3 (+3%)
AdaFuse (ours)	3.1 (+29%)	13.8 (+7%)

Table 5: Decoding latency and peak memory overhead of different dynamic adapters.

Ablation Study

To understand the contribution of individual components in AdaFuse, we conduct ablation studies focusing on the key system-level optimization: the SGMM kernel.

Method	Latency (ms/token)
Llama2-7B	2.4
MoRAL	8.5 (+254%)
MoRAL (Simple merge)	4.5 (+88%)
AdaFuse (Simple merge)	4.2 (+ 75%)
AdaFuse	3.1 (+29%)

Table 6: Ablation study for runtime decoding latency of AdaFuse.

Table 6 demonstrates the critical importance of the SGMM kernel in achieving our performance gains. Replacing the SGMM with a simple merge approach results in a substantial increase in decoding latency for AdaFuse, rising from 3.1 ms/token to 4.2 ms/token. This validates our hypothesis that kernel-level optimizations are essential for dynamic adapter efficiency.

The simple merge approach, while algorithmically equivalent, fails to exploit the parallelism opportunities inherent in our fused adapter switching strategy. We also tested this simple merge technique on MoRAL, which reduced its latency to 4.5 ms/token but still registered an 88% increase over the original Llama2-7B. These findings highlight the effectiveness of our system-algorithm co-design approach.

Conclusion

This paper addresses the critical inference latency overhead (250-950%) in dynamic LLM adapters, which we identify as a consequence of fragmented CUDA kernel calls, not computational load. We propose AdaFuse, a system-algorithm co-design featuring a token-wise pre-gated architecture. AdaFuse uses a single router at the first layer to determine expert activations for all subsequent layers, a strategy that eliminates the overhead of traditional layer-wise routing. To realize this design, we developed the SGMM CUDA kernel to perform fused adapter switching, which merges activated adapters into the backbone in a single efficient operation. Rigorous experiments validate our approach: AdaFuse achieves competitive accuracy (60.12% general, 83.58% domain-specific) while delivering a 2.4× decoding speedup over the fastest dynamic adapters. This reduces the latency overhead from a typical 250-950% to just 29% over the backbone model. Ablation studies confirm that both the pre-gated architecture and our SGMM kernel are essential to this performance. Ultimately, AdaFuse establishes a new benchmark for efficient LLM tuning, providing a practical solution that maintains model adaptability while achieving near-native inference speeds and contributing new design principles for future dynamic adapter architectures.

References

Chen, L.; Ye, Z.; Wu, Y.; Zhuo, D.; Ceze, L.; and Krishnamurthy, A. 2023. Punica: Multi-Tenant LoRA Serving. arXiv:2310.18547.

Chen, X.; Li, Y.; Cai, H.; Ma, Z.; Chen, X.; Xiong, H.; Wang, S.; He, B.; Sun, L.; and Yin, D. 2025. Multi-Agent Proactive Information Seeking with Adaptive LLM Orchestration for Non-Factoid Question Answering. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 4341–4352.

Clark, P.; Cowhey, I.; Etzioni, O.; Khot, T.; Sabharwal, A.; Schoenick, C.; and Tafjord, O. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *ArXiv*, abs/1803.05457.

Cui, X.; Lu, W.; Tong, Y.; Li, Y.; and Zhao, Z. 2025a. Diffusion-based multi-modal synergy interest network for click-through rate prediction. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 581–591.

Cui, X.; Lu, W.; Tong, Y.; Li, Y.; and Zhao, Z. 2025b. Multi-Modal Multi-Behavior Sequential Recommendation with Conditional Diffusion-Based Feature Denoising. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1593–1602.

Dou, S.; Zhou, E.; Liu, Y.; Gao, S.; Zhao, J.; Shen, W.; Zhou, Y.; Xi, Z.; Wang, X.; Fan, X.; Pu, S.; Zhu, J.; Zheng, R.; Gui, T.; Zhang, Q.; and Huang, X. 2024. LoRAMoE: Alleviate World Knowledge Forgetting in Large Language Models via MoE-Style Plugin. arXiv:2312.09979.

Feng, W.; Hao, C.; Zhang, Y.; Han, Y.; and Wang, H. 2024. Mixture-of-LoRAs: An Efficient Multitask Tuning for Large Language Models. arXiv:2403.03432.

Frantar, E.; and Alistarh, D. 2023. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. *ArXiv*, abs/2301.00774.

Frantar, E.; Ashkboos, S.; Hoefler, T.; and Alistarh, D. 2022. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. *ArXiv*, abs/2210.17323.

Gao, C.; Chen, K.; Rao, J.; Sun, B.; Liu, R.; Peng, D.; Zhang, Y.; Guo, X.; Yang, J.; and Subrahmanian, V. 2024. Higher Layers Need More LoRA Experts. arXiv:2402.08562.

Gao, L.; Tow, J.; Abbasi, B.; Biderman, S.; Black, S.; DiPofi, A.; Foster, C.; Golding, L.; Hsu, J.; Le Noac'h, A.; Li, H.; McDonell, K.; Muennighoff, N.; Ociepa, C.; Phang, J.; Reynolds, L.; Schoelkopf, H.; Skowron, A.; Sutawika, L.; Tang, E.; Thite, A.; Wang, B.; Wang, K.; and Zou, A. 2023. A framework for few-shot language model evaluation.

Gou, Y.; Liu, Z.; Chen, K.; Hong, L.; Xu, H.; Li, A.; Yeung, D.-Y.; Kwok, J. T.; and Zhang, Y. 2024. Mixture of Cluster-conditional LoRA Experts for Vision-language Instruction Tuning. arXiv:2312.12379.

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)*.

Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; De Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; and Gelly, S. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, 2790–2799. PMLR.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*.

Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S.; de las Casas, D.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; Lavaud, L. R.; Lachaux, M.-A.; Stock, P.; Scao, T. L.; Lavril, T.; Wang, T.; Lacroix, T.; and Sayed, W. E. 2023. Mistral 7B. arXiv:2310.06825.

Jiang, A. Q.; Sablayrolles, A.; Roux, A.; Mensch, A.; Savary, B.; Bamford, C.; Chaplot, D. S.; Casas, D. d. l.; Hanna, E. B.; Bressand, F.; et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C. H.; Gonzalez, J. E.; Zhang, H.; and Stoica, I. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Lester, B.; Al-Rfou, R.; and Constant, N. 2021a. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Lester, B.; Al-Rfou, R.; and Constant, N. 2021b. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Conference on Empirical Methods in Natural Language Processing*.

Li, D.; Ma, Y.; Wang, N.; Cheng, Z.; Duan, L.; Zuo, J.; Yang, C.; and Tang, M. 2024. MixLoRA: Enhancing Large Language Models Fine-Tuning with LoRA based Mixture of Experts. arXiv:2404.15159.

Li, X. L.; and Liang, P. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, abs/2101.00190.

Li, Y.; Cai, H.; Kong, R.; Chen, X.; Chen, J.; Yang, J.; Zhang, H.; Li, J.; Wu, J.; Chen, Y.; et al. 2025a. Towards AI Search Paradigm. *arXiv preprint arXiv:2506.17188*.

Li, Y.; Lyu, Z.; Zhang, Y.; Zhang, H.; Peng, T.; Xiong, H.; Wang, S.; Kong, L.; Chen, G.; and Yin, D. 2025b. S3PRank: Towards Satisfaction-oriented Learning to Rank with Semi-supervised Pre-training. *IEEE Transactions on Knowledge and Data Engineering*.

Li, Y.; Xiong, H.; Kong, L.; Wang, Q.; Wang, S.; Chen, G.; and Yin, D. 2023a. S2phere: Semi-supervised pre-training for web search over heterogeneous learning to rank data. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 4437–4448.

Li, Y.; Xiong, H.; Kong, L.; Wang, S.; Sun, Z.; Chen, H.; Chen, G.; and Yin, D. 2023b. Ltrgcn: Large-scale graph convolutional networks-based learning to rank for web search. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 635–651. Springer.

Li, Y.; Xiong, H.; Kong, L.; Zhang, R.; Xu, F.; Chen, G.; and Li, M. 2023c. Mhrr: Moocs recommender service with meta hierarchical reinforced ranking. *IEEE Transactions on Services Computing*, 16(6): 4467–4480.

Li, Y.; Xiong, H.; Wang, Q.; Kong, L.; Liu, H.; Li, H.; Bian, J.; Wang, S.; Chen, G.; Dou, D.; et al. 2023d. COLTR: Semi-supervised learning to rank with co-training and over-parameterization for web search. *IEEE Transactions on Knowledge and Data Engineering*, 35(12): 12542–12555.

Li, Y.; Xiong, H.; Zhang, Y.; Bian, J.; Peng, T.; Li, X.; Wang, S.; Kong, L.; and Yin, D. 2025c. Rankelectra: Semi-supervised pre-training of learning-to-rank electra for web-scale search. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, 2415–2425.

Li, Y.; Zhang, H.; Zhang, H.; Cai, H.; Ma, X.; Wang, S.; Xiong, H.; Ren, Z.; de Rijke, M.; and Yin, D. 2025d. Fultr: A large-scale fusion learning to rank dataset and its application for satisfaction-oriented ranking. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 5583–5594.

Li, Y.; Zhang, H.; Zhang, Y.; Cai, H.; Cai, M.; Wang, S.; Xiong, H.; Kong, L.; Yin, D.; and Chen, L. 2025e. Rankexpert: A mixture of textual-and-behavioral experts for multi-objective learning-to-rank in web search. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 4578–4589.

Li, Y.; Zhang, H.; Zhang, Y.; Ma, X.; Ye, W.; Song, N.; Wang, S.; Xiong, H.; Yin, D.; and Chen, L. 2025f. M 2 oERank: Multi-Objective Mixture-of-Experts Enhanced Ranking for Satisfaction-Oriented Web Search. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, 4441–4454. IEEE.

Lian, W.; Wang, G.; Goodson, B.; Pentland, E.; Cook, A.; Vong, C.; and "Teknium". 2023. SlimOrca: An Open Dataset of GPT-4 Augmented FLAN Reasoning Traces, with Verification.

Liao, W.; Chu, X.; and Wang, Y. 2024. TPO: Aligning large language models with multi-branch & multi-step preference trees. *arXiv preprint arXiv:2410.12854*.

Liao, W.; Jiang, P.; Lv, Y.; Xue, Y.; Chen, Z.; and Li, X. 2023. MCRLe: Multi-Modal Contrastive Representation Learning For Stroke Onset Time Diagnosis. In *2023 IEEE 20th International Symposium on Biomedical Imaging (ISBI)*, 1–5. IEEE.

Lin, S.; Hilton, J.; and Evans, O. 2022. TruthfulQA: Measuring How Models Mimic Human Falsehoods. In *Proceedings*

of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 3214–3252. Dublin, Ireland: Association for Computational Linguistics.

Liu, Q.; Wu, X.; Zhao, X.; Zhu, Y.; Xu, D.; Tian, F.; and Zheng, Y. 2023. MOELoRA: An MOE-based Parameter Efficient Fine-Tuning Method for Multi-task Medical Applications. *ArXiv*, abs/2310.18339.

Liu, Z.; and Lu, W. 2025. MDN: Modality Decomposition Network for Multimodal Recommendation. In *Proceedings of the 2025 International Conference on Multimedia Retrieval*, 871–879.

Lu, P.; Mishra, S.; Xia, T.; Qiu, L.; Chang, K.-W.; Zhu, S.-C.; Tafjord, O.; Clark, P.; and Kalyan, A. 2022. Learn to Explain: Multimodal Reasoning via Thought Chains for Science Question Answering. In *The 36th Conference on Neural Information Processing Systems (NeurIPS)*.

Lu, W.; and Yin, L. 2025. DMMD4SR: Diffusion Model-based Multi-level Multimodal Denoising for Sequential Recommendation. In *Proceedings of the 33rd ACM International Conference on Multimedia*, 6363–6372.

Luo, T.; Lei, J.; Lei, F.; Liu, W.; He, S.; Zhao, J.; and Liu, K. 2024. MoELoRA: Contrastive Learning Guided Mixture of Experts on Parameter-Efficient Fine-Tuning for Large Language Models. arXiv:2402.12851.

Mihaylov, T.; Clark, P.; Khot, T.; and Sabharwal, A. 2018. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. In *EMNLP*.

Mo, M.; Lu, W.; Xie, Q.; Xiao, Z.; Lv, X.; Yang, H.; and Zhang, Y. 2025. One multimodal plugin enhancing all: CLIP-based pre-training framework enhancing multimodal item representations in recommendation systems. *Neurocomputing*, 637: 130059.

OpenChat. 2023. ShareGPT4 Dataset. Hugging Face Datasets. Accessed: 2024-05-11.

Sakaguchi, K.; Bras, R. L.; Bhagavatula, C.; and Choi, Y. 2019. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *arXiv preprint arXiv:1907.10641*.

Snowflake AI Research Team. 2024. Snowflake Arctic: The Best LLM for Enterprise AI — Efficiently Intelligent, Truly Open. Accessed on April 26, 2024.

Talmor, A.; Herzig, J.; Lourie, N.; and Berant, J. 2019. CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge. In Burstein, J.; Doran, C.; and Solorio, T., eds., *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)*, 4149–4158. Minneapolis, Minnesota: Association for Computational Linguistics.

The Mosaic Research Team. 2024. Introducing dbrx: A New State-of-the-Art Open LLM. https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm. Accessed on April 26, 2024.

Tong, Y.; Lu, W.; Cui, X.; Mao, Y.; and Zhao, Z. 2025. DAPT: Domain-Aware Prompt-Tuning for Multimodal Fake News Detection. In *Proceedings of the 33rd ACM International Conference on Multimedia*, 7902–7911.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; Rodriguez, A.; Joulin, A.; Grave, E.; and Lample, G. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971.

Touvron, H.; Martin, L.; and et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *ArXiv*, abs/2307.09288.

Wang, J.; Zhang, Z.; He, Y.; Zhang, Z.; Song, X.; Song, Y.; Shi, T.; Li, Y.; Xu, H.; Wu, K.; et al. 2024. Enhancing code llms with reinforcement learning in code generation: A survey. *arXiv preprint arXiv:2412.20367*.

Wei, X.; Lu, B.; Zhang, X.; Zhao, Z.; Shen, D.; Xia, L.; and Yin, D. 2025. Igniting Creative Writing in Small Language Models: LLM-as-a-Judge versus Multi-Agent Refined Rewards. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 17171–17197.

Wei, Y.; Wang, Z.; Liu, J.; Ding, Y.; and Zhang, L. 2023. Magicoder: Source Code Is All You Need. *arXiv preprint arXiv:2312.02120*.

Wu, H.; Zheng, H.; and Yu, B. 2024a. Parameter-Efficient Sparsity Crafting from Dense to Mixture-of-Experts for Instruction Tuning on General Tasks. arXiv:2401.02731.

Wu, H.; Zheng, H.; and Yu, B. 2024b. Parameter-Efficient Sparsity Crafting from Dense to Mixture-of-Experts for Instruction Tuning on General Tasks. *arXiv preprint arXiv:2401.02731*.

xAI. 2024. Open release of grok-1.

Xiong, H.; Bian, J.; Li, Y.; Li, X.; Du, M.; Wang, S.; Yin, D.; and Helal, S. 2024. When search engine services meet large language models: visions and challenges. *IEEE Transactions on Services Computing*.

Yang, S.; Ali, M. A.; Wang, C.-L.; Hu, L.; and Wang, D. 2024. MoRAL: MoE Augmented LoRA for LLMs' Lifelong Learning. *arXiv preprint arXiv:2402.11260*.

Ye, Z.; Li, D.; Tian, J.; Lan, T.; Zuo, J.; Duan, L.; Lu, H.; Jiang, Y.; Sha, J.; Zhang, K.; and Tang, M. 2023. ASPEN: High-Throughput LoRA Fine-Tuning of Large Language Models with a Single GPU. *ArXiv*, abs/2312.02515.

Yu, L.; Jiang, W.; Shi, H.; Yu, J.; Liu, Z.; Zhang, Y.; Kwok, J. T.; Li, Z.; Weller, A.; and Liu, W. 2023. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. *arXiv preprint arXiv:2309.12284*.

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*.

Zhang, R.; Han, J.; Liu, C.; Gao, P.; Zhou, A.; Hu, X.; Yan, S.; Lu, P.; Li, H.; and Qiao, Y. 2023. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*.

Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E. P.; Zhang, H.; Gonzalez, J. E.; and Stoica, I. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. arXiv:2306.05685.